

AD-A252 343



HUMAN-COMPUTER INTERACTION

A Journal of Theoretical, Empirical, and Methodological
Issues of User Science and of System Design

Volume 7, Number 1
1992

DTIC
ELECTE
JUN 15 1992
S A D

This document has been approved
for public release and sale; its
distribution is unlimited.



LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
Hillsdale, New Jersey Hove and London

HUMAN-COMPUTER INTERACTION

EDITOR: Thomas P. Moran, *Xerox Palo Alto Research Center*

ADMINISTRATIVE EDITOR: Patricia Sheehan, *Xerox Palo Alto Research Center*

PRODUCTION EDITOR: Brian S. Jenkins, *Lawrence Erlbaum Associates, Inc.*

EDITORIAL BOARD:

John Anderson.....	<i>Carnegie Mellon University, Pittsburgh, PA</i>
Ruven Brooks.....	<i>Schlumberger Laboratory for Computer Science, Austin, TX</i>
John Seely Brown.....	<i>Xerox Palo Alto Research Center, Palo Alto, CA</i>
Stuart K. Card.....	<i>Xerox Palo Alto Research Center, Palo Alto, CA</i>
John M. Carroll.....	<i>IBM T. J. Watson Research Center, Yorktown Heights, NY</i>
Bill Curtis.....	<i>Software Engineering Institute, CMU, Pittsburgh, PA</i>
John D. Gould.....	<i>IBM T. J. Watson Research Center, Yorktown Heights, NY</i>
Donald E. Knuth.....	<i>Stanford University, Stanford, CA</i>
Robert E. Kraut.....	<i>Bellcore, Morristown, NJ</i>
Morten Kyng.....	<i>Aarhus University, Aarhus, Denmark</i>
Clayton Lewis.....	<i>University of Colorado, Boulder, CO</i>
Thomas W. Malone.....	<i>Massachusetts Institute of Technology, Cambridge, MA</i>
Brad A. Myers.....	<i>Carnegie Mellon University, Pittsburgh, PA</i>
Allen Newell.....	<i>Carnegie Mellon University, Pittsburgh, PA</i>
Donald A. Norman.....	<i>University of California, San Diego, CA</i>
Dan R. Olsen, Jr.....	<i>Brigham Young University, Provo, UT</i>
Gary M. Olson.....	<i>University of Michigan, Ann Arbor, MI</i>
Judith S. Olson.....	<i>University of Michigan, Ann Arbor, MI</i>
Richard Pew.....	<i>Bolt, Beranek, & Newman, Inc., Cambridge, MA</i>
Peter G. Polson.....	<i>University of Colorado, Boulder, CO</i>
James R. Rhyne.....	<i>IBM T. J. Watson Research Center, Yorktown Heights, NY</i>
William B. Rouse.....	<i>Georgia Institute of Technology, Atlanta, GA</i>
Elliot Soloway.....	<i>University of Michigan, Ann Arbor, MI</i>
Lucy Suchman.....	<i>Xerox Palo Alto Research Center, Palo Alto, CA</i>
Janet H. Walker.....	<i>DEC Cambridge Research Lab, Cambridge, MA</i>
Terry Winograd.....	<i>Stanford University, Stanford, CA</i>
Richard Young.....	<i>MRC Applied Psychology Unit, Cambridge, UK</i>

Human-Computer Interaction is published quarterly by Lawrence Erlbaum Associates, Inc., 365 Broadway, Hillsdale, New Jersey 07642.

Subscriptions (based on one volume per calendar year): Institutional, \$145.00; Individual, \$39.00. Subscriptions outside of the U.S.A. and Canada: Institutional, \$170.00; Individual, \$64.00.

Copyright © 1992, Lawrence Erlbaum Associates, Inc. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording, or otherwise, without permission from the publisher.

Permission to photocopy for internal use or the internal or personal use of specific clients is granted by *Human-Computer Interaction* for libraries and other users registered with the Copyright Clearance Center (CCC) Transactional Reporting Service, provided that the base fee of \$4.00 per copy is paid directly to the CCC, 27 Congress Street, Salem, MA 01970. 0737-0024/92 \$4.00. Special requests should be addressed to Permissions Department, Lawrence Erlbaum Associates, Inc., 365 Broadway, Hillsdale, NJ 07642.

Printed in the U.S.A.

ISSN 0737-0024

Temporal Aspects of Tasks in the User Action Notation

H. Rex Hartson

Virginia Polytechnic Institute and State University

Philip D. Gray

Glasgow University

✓ ABSTRACT

The need for communication among a multiplicity of cooperating roles in user interface development translates into the need for a common set of interface design representation techniques. The important difference between design of the interaction part of the interface and design of the interface software calls for representation techniques with a behavioral view—a view that focuses on user interaction rather than on the software. The User Action Notation (UAN) is a user- and task-oriented notation that describes physical (and other) behavior of the user and interface as they perform a task together. The primary abstraction of the UAN is a *user task*.

The work reported here addresses the need to identify temporal relationships within user task descriptions and to express explicitly and precisely how designers view temporal relationships among those tasks. Drawing on simple temporal concepts such as events in time and preceding and overlapping of time intervals, we identify basic temporal relationships among tasks: sequence, waiting, repeated disjunction, order independence, interruptibility, one-way interleavability, mutual interleavability, and concurrency. The UAN temporal relations, through the notion of modal logic, offer an explicit

Authors' present addresses: H. Rex Hartson, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061; Philip D. Gray, Department of Computing Science, 17 Lilybank Gardens, Glasgow University, Glasgow G12 8QQ, Scotland.

92-19359

92 6 10 023



CONTENTS

1. INTRODUCTION
 2. THE NEED FOR BEHAVIORAL REPRESENTATION
 3. RELATED WORK
 - 3.1. Constructional Representation Techniques
 - 3.2. Behavioral Representation Techniques
 - 3.3. Temporal Aspects
 - 3.4. Contributions of This Work
 4. INTRODUCTION TO THE UAN
 5. THE NEED FOR TEMPORAL RELATIONS
 6. TIME
 - 6.1. Events in Time
 - 6.2. Time Intervals
 - 6.3. Preceding and Overlapping
 7. TASKS AND ACTIONS
 - 7.1. Basic Definitions
 - 7.2. Instances of Tasks
 8. TIME AND ACTIONS
 - 8.1. Actions as Happenings in Time
 - 8.2. Lifetimes
 - 8.3. The Boustrophedon Argument
 - 8.4. Interruption
 - 8.5. Idle Time
 - 8.6. Periods of Activity
 9. TEMPORAL RELATIONS AMONG USER ACTIONS
 - 9.1. Sequence
 - Task Names and Levels of Abstraction
 - Grouping, Closure, and Composition of Relations
 - 9.2. Waiting
 - 9.3. Repeating Disjunction
 - 9.4. Order Independence
 - 9.5. Interruptibility
 - Uninterruptible Tasks and Preemptive States
 - Scope of Interruptibility
 - 9.6. One-Way Interleavability
 - 9.7. Mutual Interleavability
 - 9.8. Concurrency
 10. DISCUSSION
 - 10.1. How the UAN Helps With Interface Development
 - 10.2. Conclusions
- APPENDIX. MATHEMATICAL SYMBOLOGY**
-

and precise representation of the specific kinds of temporal behavior that can occur in asynchronous user interaction without the need to detail all cases that might result.

Time is nature's way of keeping everything from happening all at once.

— Unknown

1. INTRODUCTION

The great difficulty many people have in using computers is often due to a poor design of the human-computer interface. The issue is *usability*, and high usability stems from a good design. Good designs invariably depend on an ability to understand and evaluate (and thereby improve) interface designs during the development process. Understanding and evaluating designs depends, in part, on the methods used to represent the designs. Design and representation are very closely related; *design* is a creative, mental, problem-solving process, whereas *representation* is the physical process of capturing or recording the design. The need for effective representation techniques is especially important with new interface development methods that emphasize iterative refinement and involve a multiplicity of separate but cooperating roles for producing the interface. These roles include at least designer, implementer, evaluator, documenter, marketing, customer, and user. Each of these roles has its own, often different, needs for communicating—recording, conveying, reading, and understanding—an interface design. This communication need translates into the need for a common set of interface design representation techniques—the mechanism for completely and unambiguously capturing an interface design as it evolves through all phases of the life cycle. Development of a user-centered interface design necessitates that these techniques have a *behavioral view*—a view that focuses on the user rather than on the software.

The UAN has provided an answer to the need for a behavioral representation technique (Hartson, Siochi, & Hix, 1990; Siochi & Hartson, 1989). The UAN is a user- and task-oriented notation that describes physical (and other) behavior of the user and interface as they perform a task together. The primary abstraction of the UAN is a *user task*. An interface is represented as a quasi-hierarchical structure of asynchronous tasks, the sequencing within each task being independent of that in the others. User actions, corresponding interface feedback, and state change information are represented at the lowest level. Levels of abstraction hide these details and build the task structure.

The UAN has been found by many to be expressive and highly readable because of its simplicity, natural enough so that it is easily read and written with almost no training. Use within design and implementation projects has shown the UAN to be effective in conveying large and complex user interface designs from designers to implementers and evaluators. Because the UAN is task oriented, it provides a crucial articulation between task analysis and

Lewrence Enlbaum Assoc, Inc
365 Broadway
Hillsdale, New Jersey
\$39.00
NW 6/12/92



Special

A-1 21

<input checked="" type="checkbox"/>
<input type="checkbox"/>
<input type="checkbox"/>
<input type="checkbox"/>
39.00
codes
Special

In addition to the need for a behavioral view, new styles of interaction involving direct manipulation of graphical objects and icons necessitate temporal considerations. These interaction styles are more difficult to represent than the older styles of command languages and menus. User actions are asynchronous, having more complex temporal behavior than those of the old style interfaces that were largely constrained to predefined sequences. Heretofore, representation of temporal relationships has been ad hoc in the UAN. The work reported here addresses the need to identify more formally temporal relationships within task descriptions and to express explicitly and precisely how designers view temporal relationships among tasks.

We are not proposing a general theory of time or of tasks; we are merely applying some intuitive physical notions about time and the temporal properties and relationships of user tasks and computer processes. This is not a cognitive model of time (even though it refers to user's behavior), and none of our inferences or conclusions depends on features that are inconsistent with other views of time.

Out of the many logically possible temporal relationships among tasks that people carry out using computers, we have identified several that we believe to be fundamental to describing interaction (e.g., sequencing, interrupting, and interleaving). These notions are not new to designers of interactive systems, but their use has often been informal and imprecise. It is our hope that clear and precise definitions of these concepts will provide a foundation on which to reason about the temporal characteristics of interaction between users and computer systems.

A word is in order here about the structure of this article. We have used a top-down approach, often mandated by formal documentation of technical concepts. This approach has the advantage that it weaves the whole article into a connected development of concepts and definitions, defining each term before it is used. The disadvantage, however, is that this approach necessarily defers the "real content" until the end. The first five sections present introduction and motivation. The temporal relations themselves are discussed in Section 9, with Sections 6, 7, and 8 building a foundation of definitional fabric on which Section 9 is laid.

Those desiring a full logical development should read this article in the order in which it appears. A reader interested only in gaining intuitive knowledge of the temporal relations can skip Sections 6, 7, and 8; can begin with Section 9; or can read only Section 9. This reader, however, must expect to encounter many terms lacking a formal definition. As a further guide for the reader, equations in the article are numbered and set off from the text. Readers not wishing to sort out the meanings of the symbols and equations can skip all equations and still understand most of the ideas. The article is

written in a style to support this mode of reading; each equation is preceded by a prose description of what is stated in the equation.

2. THE NEED FOR BEHAVIORAL REPRESENTATION

Historically, and practically, many user interfaces have been designed by software engineers and programmers as part of the software of an interactive system. The result has been interfaces of varying quality and usability. Much work in the field of human-computer interaction has been directed toward new approaches to user interface development in hopes of improving quality and usability. From this work, it has become clear that there is an important difference between design of the interaction part of an interface and design of user interface software and that interaction design has special requirements not shared by software design. Good interaction design must be user centered. Being user centered means focusing on the behavior of a user performing tasks with the computer. To emphasize this distinction, we use the terms *behavioral domain* and *constructional domain* to refer, respectively, to the working worlds of the people who design and develop the interaction part of user interfaces and the people who design and develop the software to implement those interfaces (Hartson et al., 1990).

Most representation techniques currently used for interface software development (e.g., state transition diagrams, event-based mechanisms, window managers, software toolkits, object-oriented programming) are constructional—and properly so. Any description that can be thought of as being performed by the system is constructional. For example, a state transition diagram represents the system view, looking out at the user and waiting for an input. This diagram shows the current system state and how each input takes the system to a new state. Constructional representation techniques support the designer and implementer of the interface software but do not support design of the interaction part of the interface itself. In contrast, it is in the behavioral domain—from the user's view—that developers of the interaction part of an interface (e.g., interaction designers and evaluators) do their work. A description performed by the user (e.g., performance of a task) is behavioral. In the behavioral domain, one gets away from the software issues and into the processes that precede software design, such as task analysis, functional analysis, task allocation, and user modeling. Consequently, there is a need for behavioral representation techniques (and supporting tools) to give a user-centered focus to interface development and to serve interface developer roles. As Richards, Boies, and Gould (1986, p. 216) stated about tools for mocking up user interface prototypes, "Few of these provide an interface specification language directly usable by behavioral specialists."

With current emphasis on user-centered design (Norman & Draper, 1986),

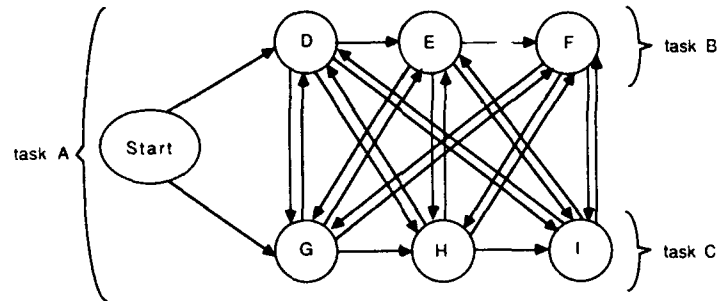
the interface development process is driven heavily by user requirements and task analysis. Early evaluation of designs is based on user- and task-oriented models (e.g., see Reisner, 1981). In fact, the entire interface development life cycle is becoming centered around evaluation of users performing tasks (Hartson & Hix, 1989; Nickerson & Pew, 1990). Thus, most interface development activity that precedes constructional design and implementation is done in the behavioral domain leading to the user task as the common element among developer roles. Behavioral representation techniques are not replacements for constructional techniques; they just support a different domain. Interaction designs represented behaviorally must still be translated into the constructional domain for interface software design and implementation. Interaction designs become requirements for the design (and implementation) of user interface software. A formal representation of interaction designs is, therefore, needed to convey these requirements, and the UAN is intended for that purpose.

3. RELATED WORK

3.1. Constructional Representation Techniques

In comparing the UAN with earlier techniques, we begin with state transition diagrams (STDs). Because STDs are a constructional representation technique, we are comparing apples and oranges, but STDs are a common basis for interface representation, and, in the absence of good oranges, designers have been known to try apples. Although STDs can be used to supplement UAN task descriptions to represent certain aspects of task transitions and interface state (Hartson et al., 1990), they cannot represent interface feedback or appearance, and the power of standard STDs to represent relationships among tasks is limited to single-stream sequential control flow. STDs theoretically can represent the other temporal relationships by representing explicitly all possible sequential control flow paths, but the result is unusable—overwhelmingly large and complex, obscuring the very sequencing structure that transition diagrams are good at showing. Two independent, asynchronous tasks must be cast together as a single entity in a synchronous model. To represent asynchronism and interleavability of two tasks, in addition to the regular state transitions within a task, each state of one is a next state of every state in the other and vice versa. For example, consider this small generic example. Suppose task A has subtasks B and C that are temporally interleavable (the user can move back and forth, suspending each one while working on the other). In the UAN this is represented as:

Task A
 $B \Leftrightarrow C$

Figure 1. State transitions diagram for task A, a simple example of interleaving.

where \Leftrightarrow means “is *interleavable* with,” a temporal relation explained in Section 9.7. For simplicity, let tasks B and C be composed of sequential steps, as shown here in the UAN:

Task B

D

E

F

Task C

G

H

I

The STD for this simplest possible example of interleaving, shown in Figure 1, would contain complicated transition conditions that depend on the real current state in each separate sequence. To include that real state information in the STD of Figure 1, one would have to replicate every possible subsequence of task B in conjunction with every subsequence of task C, producing a combinatorial explosion of states and transitions. Just as one example, if task B is really at subtask E and task C is at I, then transitions are not legal from E to G or H nor from I to D. For real tasks, such as the use of spreadsheets and text editors, the result is overwhelmingly large and complex.

It is equally important to note here that the original intention was to represent the asynchronous relationship between tasks B and C. The sequencing of B is independent of the sequencing of C, but the diagram obscures that relationship by interconnecting them.

It is clear that possible transitions between subtasks of B and subtasks of C in the just-cited example must be represented implicitly, an approach taken, for example, by Jacob (1986) and by Wellner (1989). Because direct manipulation interfaces are composed of many individual simple dialogues that interact like coroutines, Jacob divided an interface into interaction

objects, each with a separate specification based on an STD. Coroutine calls among STDs give the necessary asynchronism, suspending execution of the calling STD and remembering its current state (part of the real state discussed before). One interaction object is active at a time, and one state is current within each object.

Wellner's similar approach describes, as an example, copy-machine controls with two buttons, one for toggling through choices of paper trays and one for toggling through exposure settings. Both Jacob's and Wellner's approaches separate all states relating to paper trays from those relating to exposure. In each case, the asynchronism between the two sequences is implicit in the rules for STD operation.

It is also useful to compare the kind of concurrency that can be represented by state charts and by the UAN. The two operations in Wellner's example, selecting a paper tray and setting exposure, are represented in state charts as being concurrently available to the user. This is really *interleaving* of those operations and is what Lorin (1972) called "apparent concurrency" as contrasted with "real concurrency." In the UAN, interleaving is explicitly distinguished from real concurrency, which involves the ability of the user to do both operations simultaneously, something not addressed by state charts. To the designer the difference may be just an implementational detail, but to the user it is significant.

Most representation techniques used with user interface management systems are constructional, including STDs, asynchronous STDs, and state charts. There are also event handlers (Green, 1985; Hill, 1987), which describe system actions (e.g., invocation of a computational procedure) in response to events resulting from user actions. Event handlers introduce an object-oriented flavor and, therefore, are even better suited for representing asynchronism. They have more expressive power than STDs (Green, 1986) but suffer in comparison, as do most object-oriented approaches, when there is a need to visualize or trace sequences of user operations.

3.2. Behavioral Representation Techniques

All representation techniques in the previous section are constructional. The UAN is task oriented and behavioral, so it does not compete with STDs, for example. Both kinds of techniques are necessary for interface development, but behavioral methods are needed specifically for interaction design.

Grammatical representations using Backus Naur Form (e.g., Syngraph; Olsen & Dempsey, 1983) tend to be behavioral because they describe expressions that come from the user, but they are difficult to write and understand. Also, like standard STDs, grammars typically do not represent interface feedback, do not represent the appearance of the interface, and are not suitable for asynchronism. Multiparty grammars (Shneiderman, 1982), an interesting extension to production-rule-based techniques, do support

direct association of interface feedback with user inputs. Multiparty grammars, however, are not easily adapted to the variety of user actions in direct manipulation interfaces.

One behavioral technique that has long been used both formally and intuitively involves scenarios (or storyboarding) of interface designs. This technique is effective for revealing an early picture of interface appearance and behavior. But, because a scenario is an example of the interface, it cannot represent the complete description of the user's behavior while interacting with the computer. Peridot (Myers, 1987) is based on specification of interfaces by demonstration—The user carries out the actions of the scenarios. The use of inference and confirming dialogue solves the problem of generalizing a design from specific instances of interaction. This approach is novel, but Peridot produces program code directly with no intermediate representation that can convey interface designs or behavior or that can be analyzed.

Most other behavioral techniques are generally task oriented, including the GOMS model (Card, Moran, & Newell, 1983); the Command Language Grammar (CLG; Moran, 1981); the Keystroke-Level Model (Card & Moran, 1980); the Task Action Grammar (Payne & Green, 1986); and the work by Reisner (1981), Kieras and Polson (1985), and Sharratt (1990). Design of interactive systems, as with most kinds of design, involves an alternation of analysis and synthesis activities (Hartson & Hix, 1989). Most of the models just mentioned were originally oriented toward analysis; that is, they were intended to represent an existing design in order to evaluate usability by predicting user performance, rather than to capture a design as it is being developed. On the other hand, synthesis includes activities that support the processes of creating a new interface design and capturing its representation. The UAN shares the task orientation of these other behavioral models but is more synthesis oriented, because it was created specifically to communicate interface designs to software engineers and implementers. In practice, most techniques mentioned before can be used to support synthesis as well but typically do not represent the direct association of feedback and state with user actions. Also, many of these models—the GOMS, CLG, and keystroke in particular—are models of expert error-free task performance in contiguous time (without interruption, interleaving of tasks, and without considering the interrelationships of concurrent tasks), not suitable assumptions for the synthesis-oriented aspects of interface design.

3.3. Temporal Aspects

The phenomena with which we are concerned in this article—user actions during interaction with computer systems—are similar to computer-based processes and to human cognitive behavior in that they all exhibit

sequentiality through time. That is, we can measure the amount of time taken for their execution, perhaps identify beginning and endpoints for their duration, and describe temporal relations among them. It should not be surprising, therefore, to find formalisms similar to our own for describing and reasoning about the temporal aspects of such processes and behavior.

Temporal logics have been developed for a number of applications in computer science, cognitive science, and artificial intelligence, among which are:

- reasoning about concurrent systems, including program verification (Barringer, 1985), operating systems, and very large scale integration design (Moszkowski, 1986);
- reasoning about database updates (Kowalski & Sergot, 1986);
- systems for temporal logic programming (Hale, 1987);
- building theories and automated systems to model human planning behavior (Allen, 1983, 1984; McDermott, 1982); and
- natural language understanding systems (Kahn & Gorry, 1977).

Two basic approaches to handling time are employed in these applications. One approach, employed largely for natural language understanding and temporal logic programming, uses a tense logic with modal operators that express the temporal dependencies of the truth values of propositions. However, where the goal is to describe the temporal attributes of events and processes, the truth value of propositions need not be treated as time dependent. This second approach, which we adopt in this article, models time as entities or attributes of entities that are then described using first-order predicate calculus.

Of these applications, the work closest to ours is that of Allen, which is concerned with describing and automating reasoning about human planning and conversation. Using first-order logic, Allen identifies 13 basic temporal relations among events and processes, such as "before," "during," and "overlaps," among which are the relations with which we are concerned. The main difference between Allen's theory and our own lies in his adoption of intervals of time, rather than time points, as primitive, with a consequent effect on the handling of the interruptibility of actions; this is discussed further in Section 6.2. As Allen has noted, however, it is possible to recast his theory with time points as primitives.

Constructional interface design models have not used temporal logic up to now, but that is not to say that they have failed to capture temporal aspects of interface behavior. Transition networks are capable of modeling sequential temporal ordering but are not capable of representing the temporal relation-

ships within asynchronous and concurrent interaction (Green, 1986). Production systems, based on sets of event-response rules, have been proposed as a means of capturing the more complex temporal relations among events in modern interactive systems (Duce, 1985; Hill & Hermann, 1989). However, the complex temporal relations that they are capable of capturing are hidden in the implicit semantics of rule selection, and hence these relations are neither explicitly expressed nor capable of being reasoned about.

Cardelli and Pike's (1985) Squeak, based on cooperating sequential processes, is a language for describing interfaces that exhibit concurrency. Thus, an interface is described in terms of a set of processes, each of which accepts events as input and generates events as output. Processes communicate with one another by the transmission of an output event from one process serving as the input to another. Unlike other constructional interface models, a formal semantics for Squeak has been defined in which there is explicit reference to the passage of time, which is used to express control flow among processes in terms of null actions during which time units elapse. However, no attempt is made, as with Allen and others, to model the temporal aspects of the system based on a theory of time and temporal relations.

The models of interaction described earlier are all constructional, in the sense that they represent interaction from the system's viewpoint. Mechanisms for handling control and communication from a programming point of view are not likely to capture all the temporal relations that exist among actions from a user's point of view. For example, the difference between interleaved processes and concurrent processes may be an implementational detail for a constructional description of the interface and hence, as in Squeak, does not appear in the abstractions of the language. As mentioned in Section 3.1, however, interleaved and concurrent actions are significantly different from a user's point of view. A behavioral description of the interaction must be able to express the difference and should be built on a theory of temporal relations among user actions that explicates the difference.

It should be emphasized that we are concerned here with the temporal aspects of user activity, not with the user's perception of temporal relations among these actions. Thus, recent work on the influence of the perception of time and the efficacy of human reasoning about temporal relations among processes (Decortis & De Keyser, 1988) is not relevant to our concerns.

3.4. Contributions of This Work

The need for synthesis-oriented behavioral techniques for interaction design representation was motivated in Section 2. In addition, designers need a precise framework in which to think about, discuss, and represent constraints on relative timing among asynchronous tasks. More motivation for

temporal relations is presented in Section 5. Sections 3.1 through 3.3 show that nothing already exists to fill these needs.

The UAN, as described in this article, does meet the need for a synthesis-oriented behavioral representation technique with temporal relations. The UAN is the only representation technique that provides synthesis-oriented, behavioral representation of tasks in interface designs, independent of implementation concerns, and with the temporal relations necessary to represent today's asynchronous interface designs. Further, design representation is not about actions a user makes so much as it is about actions a user *can* make. In an asynchronous environment, it is especially important to be able to represent specific kinds of behavior that *can* occur without having to detail all the cases that might result. The UAN temporal relations, through the notion of modal logic, offer an explicit and precise representation of what tasks can be interrupted, interleaved, and performed concurrently.

4. INTRODUCTION TO THE UAN

Use of the basic UAN, without emphasis on temporal aspects, is introduced briefly here by way of example. Figure 2, adapted from (Hartson et al., 1990), summarizes many of the UAN symbols, with only the temporal relations needed for the examples in this section. These symbols for the basic physical user actions are at the lowest level of abstraction and are suggested symbols in the sense that the UAN is an open notation, often adapted and extended by interface designers. Tasks composed of these actions are named, and the names are used as references to the task descriptions in order to build up levels of abstraction in a task structure as described in Section 9.1.

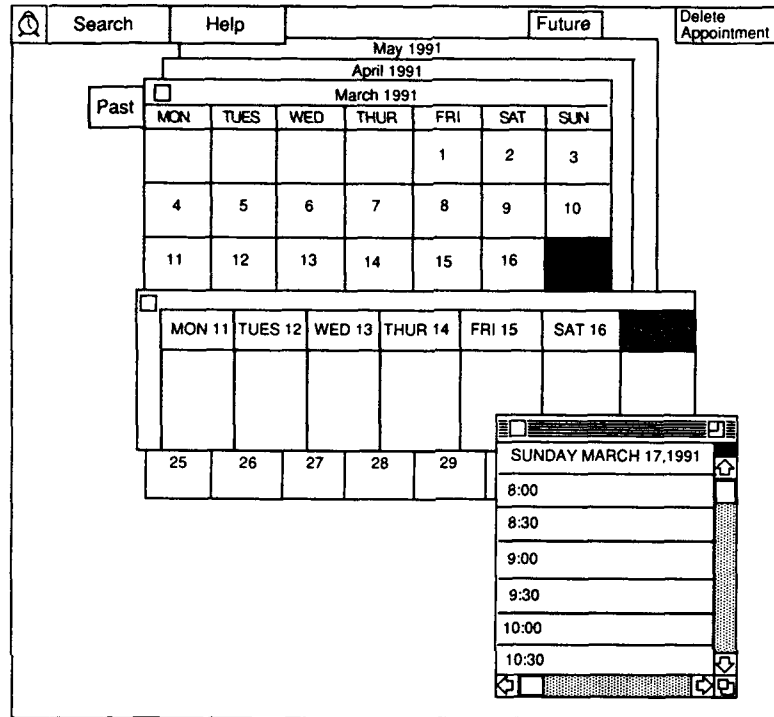
As an example, consider a hypothetical Calendar Management System (CMS) that maintains appointments in a small database. The main interface object is the display of a calendar with views for day, week, and month (as shown in Figure 3) through which the user can navigate. The paradigm for adding, modifying, or deleting an appointment is simple and analogous to the paper calendar: Find the correct day (via day, week, and/or month views) and hour and type into the appointment spaces. There are also commands for searching the calendar and for help information. The highest level UAN task description for using CMS might be as shown in Figure 4. Each time CMS is used, the user makes one choice from among its basic functions: `access_appointment`, `add_appointment`, `update_appointment`, `delete_appointment`, or `establish_alarm`. This choice is represented in Figure 4 by the disjunction symbol (`|`). The task of selecting and executing one basic function can be performed any number of times, represented in Figure 4 by the `*` symbol. Task analysis is the process that reveals the need for the basic tasks in Figure 4, but details of the methods for performing those tasks may not be

Figure 2. State transitions diagram for task A, a simple example of interleaving.

Action	Meaning
-	Move the cursor
[X]	The context of object X, the "handle" by which X is manipulated
-[X]	Move cursor into context of object X
-[x,y]	Move the cursor to (arbitrary) point x,y outside any object
-[x,y in A]	Move the cursor to (arbitrary) a point within (relative to) object A
-[X in Y]	Move to object X within object Y (e.g., [OK_icon in dialogue_box])
[X]-	Move cursor out of context of object X
v	Depress
^	Release
Xv	Depress button, key, or switch called X
X^	Release button, key, or switch X
Xv^	idiom for clicking button, key, or switch X
X"abc"	Enter literal string, abc, via device X
X(xyz)	Enter value for variable xyz via device X
()	Grouping mechanism
*	Iterative closure, task is performed zero or more times
+	Task is performed one or more times
{ }	Enclosed task is optional (performed zero or one time)
OR,	Disjunction, choice of tasks (used to show alternative ways to perform a task)
:	Separator between condition and action or feedback
Feedback	Meaning
!	Highlight object
!-	Dehighlight object
!!	Same as !, but use an alternative highlight
!-!	Blink highlight
(!-!) ⁿ	Blink highlight n times
@x,y	At point x,y
@X	At object X
@x,y in X	At point x,y in (relative to) object X
Display(X)	Display object X
Erase(X)	Erase object X
X>~	Object X follows (is dragged by) cursor
X>>~	Object X is rubber banded as its follows cursor
Outline(X)	Outline of object X

known at first. (The UAN supports development of the design in any direction of abstraction—top down, bottom up, and inside out.) Later, the subtasks of the `access_appointment` task might be described in the UAN as shown in Figure 5.

To access an appointment, this task description specifies that the user does any number of `search`, `access_month`, `access_week`, and `access_day`

Figure 3. Typical user's view of the CMS.**Figure 4.** Manage_calendar task description.

Task: manage_calendar

(access_appointment
 | add_appointment
 | update_appointment
 | delete_appointment
 | establish_alarm)*

tasks followed by a single `access_time_slot` task (time slots being containers of appointments). Figure 6 shows further details of the `access_month` task. The `access_week` and `access_day` tasks are very similar.

The first subtask, `select(any_month)`, allows the user to make the month level the current view level and is instantiated by substituting a specific month

Figure 5. Access_appointment task description.

Task: access_appointment

```
(search
| access_month
| access_week
| access_day)*
access_time_slot
```

Figure 6. Access_month task description.

Task: access_month

```
(select(any_month)
| move_forward_by_month
| move_backward_by_month)*
```

on the screen for “any month” and using a parameterized task description¹ for select (see Figure 7). Because the **select(object)** task description is composed of primitive user actions, it is more detailed and contains (among other possibilities) columns for user actions, interface feedback, and interface state.

The symbols are explained here in approximately the order of their appearance. In the first column, the ~ means to move the cursor, and square brackets, [and], around an object denote the context of that object. Thus, ~[X] means to move the cursor to the context of X. The context of an object is that by which the object is manipulated, which is often the object itself, or it can be, for example, a circumscribed rectangle or “grab handles” such as those used to manipulate line objects in a drawing application. In Figure 7, the item contained in square brackets denotes any arbitrary object icon, but the modifying condition (~!) further specifies that the object icon must not be already highlighted. Therefore, the first line in the task reads: Move the cursor to an unhighlighted object icon and depress the mouse button (Mv). The corresponding feedback, shown in the middle column, is highlighting of the object icon (object_icon!). For this task, selection is defined (elsewhere) to be from a mutually exclusive set of object icons. The feedback also indicates that any other object icon already highlighted is now unhighlighted

¹ We have used an interaction style similar to that of the Macintosh in our examples. Macintosh is a registered trademark of Macintosh Laboratories. The UAN is not limited to the Macintosh, and it is not oriented toward any one specific graphical direct manipulation style. However, we have taken advantage of the popularity of the Macintosh desktop concept to illustrate use of the UAN.

Figure 7. Select (object) parameterized task description.

Task: select(object)		
User Action	Interface Feedback	Interface State
~[object_icon'!] Mv	object_icon'!, ∀object_icon'!: object_icon'!	selected = object
Mv		

(object_icon'!: object_icon'!).² The resulting interface state (selected = object), shown in the third column, defines the set of selected items to contain exactly the one object whose representing icon is highlighted. This implies that any previously selected objects are now unselected and makes explicit the difference between selection of an object and highlighting of the icon that represents the object. The **select** task is completed in the last line of the task description by releasing the mouse button (MΛ).

Returning to the **access_month** task description in Figure 6, the first step to **select(any_month)** makes the month level the current level in navigating the calendar, and the user can then move forward or backward by month. When the user desires to navigate at the week (or day) level, this is accomplished by performing the **access_week** (or **access_day**) task that starts with **select(any_week)**, or **select(any_day)**, causing the current level to be the week (or day) level. The overall design for navigation requires access to all levels, supported by a design decision to keep at least one instance (default is current instance) of month, week, and day on the screen at all times.

The task of accessing a time slot is shown in Figure 8. The **access_time_slot** task in Figure 8 begins with a precondition, called a *condition of viability*, that means the view level for navigation must be at the day level or the user cannot perform this task. This precondition is met by performing the **access_day** task, either by itself or as part of the **access_appointment** task that precedes the **access_time_slot** subtask, as shown in Figure 5.

The task of adding a new appointment is described in Figure 9. Task transaction diagrams (Hartson et al., 1990), STDs among tasks as states, are a useful representation technique to supplement the UAN. Navigation within the CMS provides a good example where clarity is added by a task transition diagram, as shown in Figure 10.

² For simplicity, we ignore the more complex reality of the CMS that requires consideration of a containment relation. For example, a month can be selected without a week or day, but selecting a week also selects the containing month and so on.

Figure 8. *Access_time_slot* task description.

Task: *access_time_slot*

view_level = day:
 ((scroll_up | scroll_down)*
 select(any_time_slot)

Figure 9. *Add_appointment* task description.

Task: *add_appointment*

access_appointment
edit_appointment

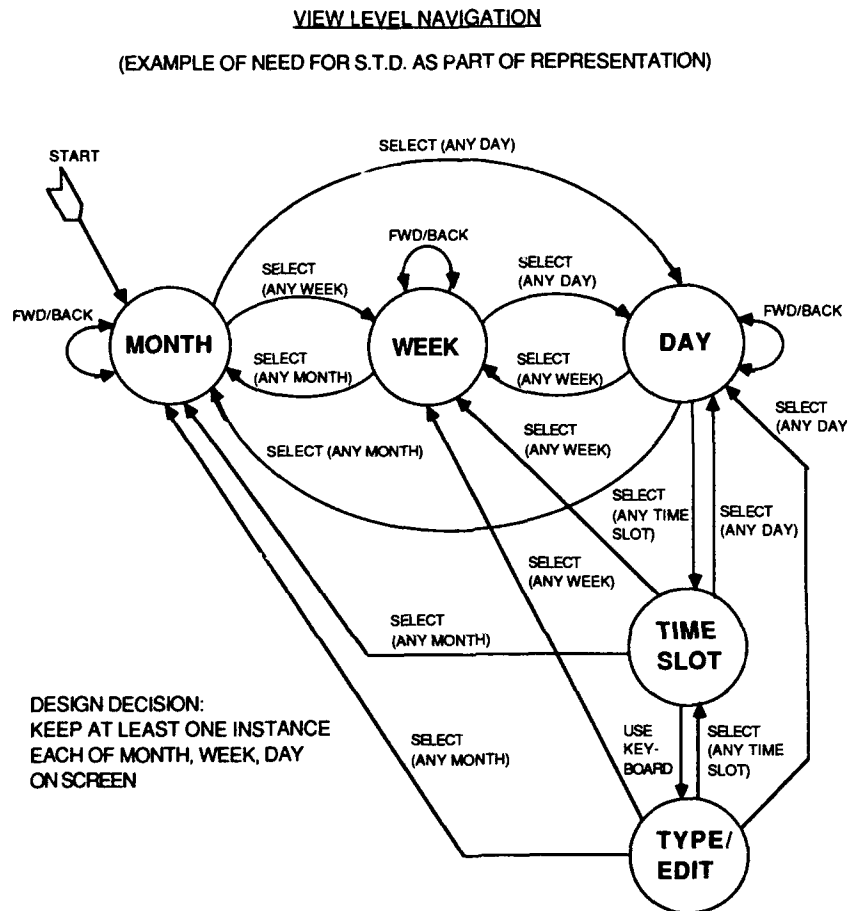
The task of establishing the alarm, to notify the user later when an appointment is impending, is described in Figure 11. The condition of viability in the first line ensures that there is a current appointment (or at least a specific time) with which to associate the alarm. The second line establishes the association of an alarm with the appointment. The third line is a matter of using a dialogue box to set parameters such as alarm lead time (how long in advance of an appointment to sound the alarm). *This dialogue box is also the means to express standing orders for alarms (such as every week at this day and time).*

The *set_alarm* task to associate an alarm with an appointment (invoked in the second line of Figure 11) is accomplished by dragging a copy of the alarm icon from the upper left-hand corner of the screen (see Figure 3) to the time slot of the appointment. The *set_alarm* task is detailed in Figure 12.

The first line of Figure 12 contains a condition of viability for the whole task. The first line of feedback (opposite *MV*) shows that the alarm icon is to be highlighted, if it is not already so. The next line of feedback shows the alarm icon to be an element of a mutually exclusive set of command icons, causing any other already selected icon in the set to be unhighlighted (and unselected) when the mouse button is depressed over the alarm icon. The user action *-[x,y]** describes movement of the cursor to various arbitrary points over the screen, on the way to the appointment. Feedback for this action shows that the icon itself stays in place at the top of the screen while the outline of a copy of the icon gets dragged away. The feedback for the action of releasing the mouse button (*MA*) indicates that the copy of the alarm icon is affixed to the appointment display at a specific point (*x'*, *y'*) relative to the appointment itself.

An interesting part of the temporal nature of a task is the phrasing or chunking that occurs among user actions (Buxton, 1983). For example, the

Figure 10. Task transition diagram depicting navigational possibilities among some CMS tasks.



task description of Figure 12 clearly and visually delineates the part of the task performed while the mouse button is depressed as everything that occurs in the task description between **MV** and **MA**.

As another example of phrasing, consider the task of multiple icon selection with the Shift key, as shown in Figure 13. Here the interval over which the Shift key is depressed is a "phrase" that spans the selection (and/or deselection) of as many icons as desired and can easily be identified visually in the task description.

5. THE NEED FOR TEMPORAL RELATIONS

Temporal relations were not emphasized in Section 4, which introduced the basic UAN. We now begin to discuss the introduction of temporal relations,

Figure 11. Establish_alarm task description.

Task: establish_alarm
 view_level = time_slot:
 set_alarm
 set_alarm_parameters

Figure 12. Set_alarm task description.

Task: set_alarm		
User Action	Interface Feedback	Interface State
<hr/>		
view_level = time_slot:		
-[alarm_icon]		
Mv	alarm_icon-!: alarm_icon!, ∀cmd_icon'!: cmd_icon'!	selected = alarm_ command
-[x,y]*	outline(copy(alarm_icon)) > -	
-[appointment_icon]	outline(copy(alarm_icon)) > -, appointment_icon!	
MA)	display(copy(alarm_icon)) @x',y' in appointment icon	

summarized in Figure 14 for reference in this section, into the UAN for use in task descriptions. Formal definitions of the temporal relations are given in Section 9.

The question of temporal aspects enters into the user interface design process when the relative timing of tasks is considered. The easiest case for the designer is often the most constraining for the user. For example, the designer of a sequence requires completion of one task before another is begun. The CMS task description in Figure 9 illustrates a sequence. The user must complete the `access_appointment` task before beginning the `edit_appointment` task. The two tasks cannot be active at the same time. However, users often wish to interrupt a task and, while they are thinking of it, perform another task, later resuming the original one. A major purpose of asynchronous direct manipulation interaction styles is to support this kind of interleaved user task behavior. It follows that there is a need for a behavioral way to represent the possibility of interleaving on the part of the user. This need is met by the interleavability relation, which is used to connect these kinds of tasks in UAN task descriptions.

Most design representations leave this question of intertask temporal relationships implicit, if not ambiguous or undefined. Such specifications often lead to arbitrary design on the part of the interface software designer or implementer. For example, in designing for the task of adding a new appointment to the calendar, a designer may look to the interface toolkit for

Figure 13. Multiple_icon_selection task description.

Task: multiple_icon_selection		
User Action	Interface Feedback	Interface State
(Sv		
-[file_icon]		
Mv	file_icon-!: file_icon!,	selected = selected \cup file
	file_icon!: file_icon -!	selected = selected - file
MA		
S ⁺) ⁺		

Figure 14. Summary of UAN temporal relation symbols.

Temporal Relation	UAN Symbology	Meaning
Sequence	A B	Tasks A and B are performed in order left to right, or top to bottom
Waiting	A (t > n) B	Task B is performed after a delay of more than n units of time following task A
Repeating disjunction	(A B)*	Choice of A or B is performed to completion, followed by another choice of A or B, and so on
Order independence	A & B	Tasks A and B are order independent (order of performance is immaterial)
Interruptibility	A \rightarrow B	Task A can interrupt task B
One-way interleavability	A \rightarrow B	Task A is one-way interleavable with B (A can interrupt B and execute, but not vice versa)
Mutual interleavability	A \leftrightarrow B	Task A and task B are (mutually) interleavable
Concurrency	A + B	Task A and task B can be performed concurrently

an appropriate "widget." It could be reasonable to the designer to use a preemptive style dialogue box (Thimbleby, 1990), requiring the user to enter information for the appointment before moving on to the next task. In contrast, a user may seek information from an existing appointment while in the midst of creating a new appointment. Or a user may wish to create two or more related appointments at once, or to set the alarm while still creating an appointment. Good interface design suggests that the designer will at least allow the user to close the dialogue box without completing the associated data

entry task and that any information entered so far will be retained. A good design might also provide copy-and-paste operations for moving information from one appointment to the other. But the user might still be left with the responsibility of closing one task and opening the other and often must use human working memory to carry certain information from one task context to another. From the user view, there is a task interruption, but the design does not support it well. Proper evaluation and design iteration will lead to a better design, but temporal relations in the behavioral representation techniques can help in two ways. First, temporal considerations might be in the design but cannot be explicit in its representation without temporal relations. The UAN temporal relations allow the designer to declare explicitly the temporal relationships among the tasks. Second, treatment of temporal aspects in this context is ad hoc, whereas temporal relations in the UAN help the designer to think a priori about temporally related design issues.

In retrospect, many UAN temporal relationships may appear deceptively obvious, but without them it is very difficult to discuss important asynchronous aspects of interface designs with precision and to distinguish among temporal alternatives within a design. In the next section, we begin to develop the formalization of temporal relations in behavioral interface representations. As mentioned at the end of Section 1, those interested in just an intuitive understanding of the temporal relations can skim or skip over Sections 6, 7, and 8.

6. TIME

In what follows, we take as given that our universe of discourse contains *time*, which is a one-dimensional quantity, made up of points, where each point is associated with a value. The points are ordered along the dimension by their values. The common concepts of later and earlier correspond to larger and smaller values of time, respectively. The view of time taken here is compatible with the traditional psychological, thermodynamic, and cosmic views of time (Hawking, 1988).

Nothing we say in this article depends on whether this quantity is discrete or continuous. User behavior certainly occurs in continuous time. At the lowest level, most corresponding computer events occur in discrete time—Continuous user inputs are sampled in the hardware, and outputs are subject to timing constraints (e.g., a system clock). Resolution of time, however, is usually sufficiently fine so that the difference in views from user to computer is insignificant.

An example of a case in which sampling resolution does make a difference is seen in a Macintosh interface when using multiple display monitors, for example, with one on top of the other. Depressing the mouse button within

the menu bar causes the corresponding menu display. With a single monitor, a user cannot move the cursor above the menu bar. However, the second monitor can provide screen space above the normal application display. In this configuration, moving up to the bar, and beyond into the second screen above, causes the menu to disappear. It is possible, though, to move up through the bar fast enough so that the cursor position is not sampled within the bar. In this case, the menu remains displayed, even though the cursor could not have gotten above the bar without passing through it. Fortunately, examples such as this are more oddities of timing than real interface problems.

The rest of Section 6 is devoted to the fundamental notions that events happen in time and that intervals of time occur and can be compared to determine if one precedes another or if two or more intervals overlap in time.

6.1. Events in Time

Things that happen in the world (i.e., events) can be thought of as happening *in* time; that is, each event can be associated with a set of points in time so that it is possible to answer questions such as: "Given a point in time, t , and an event, e , is e Happening at t ?" Formally, where t is a point in time and e is an event, consider a binary relation H such that:

$$e H t \leftrightarrow \text{event } e \text{ is Happening at time } t \quad (1)$$

Note that the right-hand side of this definition involves an appeal to the physical world, and thus it is a *postulate* of the model that the right-hand side can be evaluated. (The reader is referred to the Appendix for an explanation of mathematical notation used in the equations and elsewhere in this article.)

6.2. Time Intervals

An interval of time is an ordered set defined by an ordered pair of two points in time. Thus, an interval of time, T , denoted by $[t_1, t_2]$ is defined:

$$T = \{t \mid (t \geq t_1) \wedge (t \leq t_2)\} \quad (2)$$

We define two projection functions on intervals, B and E , to extract their Beginning and Ending points. Where T is the interval $[t_1, t_2]$:

$$B(T) = t_1 \quad (3)$$

$$E(T) = t_2 \quad (4)$$

Our adoption of time points as primitives, and the definition of intervals in terms of them, is in contrast to Allen's theory (see Section 3.3). Allen argues that the use of points of time as a primitive leads to certain semantic difficulties, particularly in handling change over time. Thus, if an action, say selecting a menu item, is defined in terms of its temporal endpoints, and time is continuous, then there must exist a time at which the user is neither selecting nor not selecting the menu item. As we argued in the previous section, however, we are not committed to treating time as continuous for the purposes of modeling user actions. Furthermore, as a consequence of taking intervals as primitives, Allen is forced to introduce separate concepts of event (indivisible through its defining interval) and process (interruptible during its defining interval). Our approach avoids this problem, resulting in an ontology containing only one type of entity for actions and an account of interruptibility with greater explanatory power (see Section 9.5).

6.3. Preceding and Overlapping

Two important relations between intervals are Precedes and Overlaps, denoted respectively by P and O. Where T_1 and T_2 are intervals:

$$T_1 P T_2 \Leftrightarrow \forall i, j ((t_i \in T_1) \wedge (t_j \in T_2)) \supset (t_i < t_j) \quad (5)$$

$$T_1 O T_2 \Leftrightarrow \exists t ((t \in T_1) \wedge (t \in T_2)) \quad (6)$$

The following section formalizes the concepts of task and user action, as used in the UAN. Then Section 8 relates user actions to time, setting the stage for the development of UAN temporal relations in Section 9.

7. TASKS AND ACTIONS

The primary abstraction of the UAN is the task. A human-computer interface is represented as a quasi-hierarchical structure of asynchronous tasks, the sequencing within each task being independent of that in the others. Each task is, in turn, represented in a notation describing user actions and interface feedback, offering a structured way to describe the cooperative performance of a task between a human user and a computer system.

The UAN was originally created to provide a pragmatic and effective means for conveying interface design ideas from designers to implementers and evaluators. It is a goal of this article to be more precise about the concepts of task and user action, which were not formally defined in the original UAN. Additionally, we wish to make the connection between user actions and time.

7.1. Basic Definitions

The basic concepts of UAN are those of task, action set, and user action. With the inclusion of temporal relations, a UAN task is an ordered triple:

$$\text{task} = \langle \text{action set, temporal relation set, application function} \rangle \quad (7)$$

The elements of the temporal relation set, when applied by the application function, specify the temporal relationships among actions in the action set.

A user action is either a primitive user action or a task:

$$\text{user action} = \text{primitive} \mid \text{task} \quad (8)$$

The action set of a task, α , is the union of all user actions mentioned in the description of α and is obtained by applying the projection function, $A(\alpha)$, to the triple of Equation 7.

The definition of *task* in Equation 7 is recursive in that the elements of the action set may themselves be tasks via Equation 8. The primitives of the UAN, into which all tasks may be decomposed, are simply those actions which, by definition, are not further decomposed; these include basic physical operations by the user on input devices (e.g., cursor movement, mouse button press and release, keypresses). Task descriptions can also include memory, cognitive, perceptual, and decision-making user actions (Sharratt, 1990), but they are not discussed here. The boolean function, $\text{prim}(\alpha)$, is used to determine if an action, α , is primitive:

$$\begin{aligned} \text{prim}(\alpha) &= \text{true, if } \alpha \text{ is a primitive user action;} \\ &= \text{false, otherwise} \end{aligned} \quad (9)$$

In the following sections, the contents of the latter two elements of the task (viz., the temporal relation set and the application function) are discussed in relation to user actions. It should be noted that task descriptions using the UAN include annotations referring to feedback, display state, and communication with the application. Although these are essential in determining the adequacy of a design specified by means of the UAN, these annotations are not germane to the issues discussed in this article.

7.2. Instances of Tasks

The UAN uses the names of actions (primarily tasks in this context) as intensional design-time references to extensional run-time *instances* or invocations of those tasks. The intensional descriptions specify constraints on

temporal *possibilities* for extensional instantiations of the tasks within a specific performance of the containing task. We use the term *instance* as needed for clarity. However, the terms *task* and *action* and related terms are often sufficient to refer to their instances, unless it is important to make the distinction.

8. TIME AND ACTIONS

8.1. Actions as Happenings in Time

Instances of user actions are events in time, and thus we wish to apply the relation H to them, where α is an instance of a user action and t is a point in time:

$$\alpha H t \leftrightarrow \text{action } \alpha \text{ is Happening at time } t \quad (10)$$

Again, it is postulated that the right-hand side can be evaluated for any user action, α . Equation 10 is fundamental in that it relates user actions to time.

8.2. Lifetimes

An instance of a user action, α , has a Lifetime, denoted $L(\alpha)$, which is the interval spanning just those times that satisfy H :

$$L(\alpha) = [B(L(\alpha)), E(L(\alpha))] \quad (11)$$

where the Beginning of the Lifetime, $B(L(\alpha))$, is the least point in time such that the action instance is happening at that time:

$$B(L(\alpha)) = t \ni ((\alpha H t) \wedge \neg \exists t'((t' < t) \wedge (\alpha H t'))) \quad (12)$$

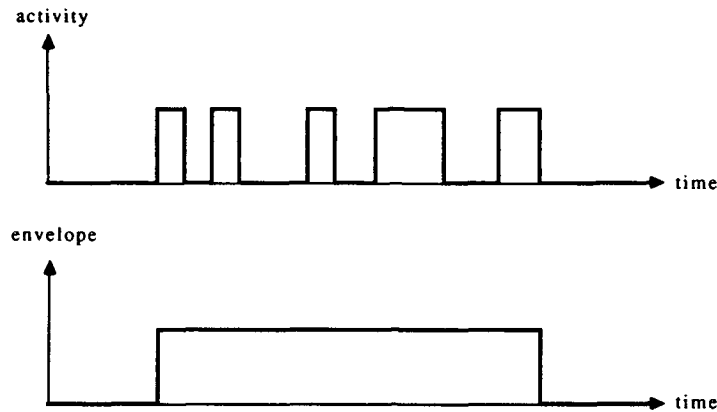
and the End of the Lifetime, $E(L(\alpha))$, is the greatest point in time such that the action instance is happening at that time:

$$E(L(\alpha)) = t \ni ((\alpha H t) \wedge \neg \exists t'((t' > t) \wedge (\alpha H t'))) \quad (13)$$

8.3. The Boustrophedon Argument

Given these definitions, a user action (primarily a task in this context) need not be happening at all times during its lifetime; there may be times of inactivity as well as times of activity. The graph of activity versus time, which we call an *activity waveform diagram*, is a boustrophedon (alternating rectangu-

Figure 15. The boustrophedon activity waveform and its envelope, the lifetime of the task instance.



lar) waveform. By the previous definition, however, the lifetime of an instance of a task is the “envelope” of the corresponding boustrophedon waveform, as shown in Figure 15.

8.4. Interruption

One way that a task can become inactive is due to interruption by another task. An interruption occurs when the user and system activities of one task are suspended before the end of the task’s lifetime and the activity of another task is begun in its place. Task interruption usually occurs due to actions initiated by the user, but they can also be the result of system-initiated actions (e.g., to update a clock or announce the arrival of electronic mail).

8.5. Idle Time

Another way that a task can be inactive is due to idle time, when neither user nor system is doing anything significant to this task. All tasks are decomposable into primitive physical user and system actions. There are natural lulls between physical actions—times between keystrokes and pauses to see, to think, or to get a cup of coffee. These lulls correspond to inactive periods in the activity waveform diagram but, by the boustrophedon argument, are part of the lifetime of the task.

8.6. Periods of Activity

Formally, a period of activity, π , of a task, α , is an interval such that α is happening at all times in the interval:

$$\pi(\alpha) = [t_1, t_2] \ni \forall t_i ((t_1 \leq t_i \leq t_2) \supset (\alpha \text{ H } t_i)) \quad (14)$$

The lifetime of a task, then, contains one or more periods of activity. As just noted, a period of activity of an instance of a task continues until it is terminated by interruption from another task or by inactivity within itself.

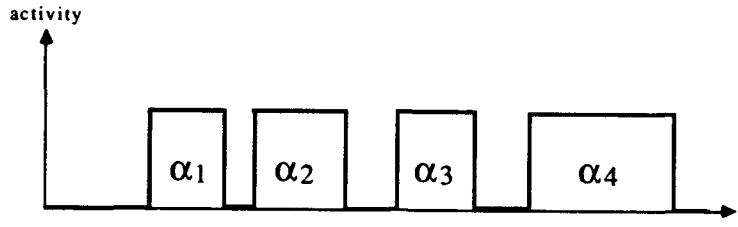
9. TEMPORAL RELATIONS AMONG USER ACTIONS

In this section, several temporal relationships among user actions are identified and formally represented. In its simplest form, each relationship is represented as a binary relation between two user actions: $\alpha_1 R \alpha_2$. In this context, it is not very useful to regard these relations as mapping operators, to think of giving an α_1 in the domain of R and yielding an α_2 in the range. As mappings, the relations described here are usually not total and not functional and are often many-to-many. Rather, it is better to think of these relations as algebraic combining operators. If user actions α_1 and α_2 are related by R , $\alpha_1 R \alpha_2$, it means that they bear a certain temporal relationship within a task, and one can perform a kind of abstraction by combining them; that is, apply R to α_1 and α_2 and, by closure (see Section 9.1), get a new user task, $\alpha_3 = R(\alpha_1, \alpha_2)$.

The most basic temporal relationships we have identified are:

- sequence
- waiting
- repeated disjunction
- order independence
- interruptibility
- one-way interleavability
- mutual interleavability
- concurrency

The set of temporal relations in a task definition defines a set of constraints (or, perhaps, the relief of constraints) among the elements of the action set. For example, if two tasks are related by a sequence, the temporal possibilities for their performance are completely constrained. On the other hand, if the same tasks are related by mutual interleavability, they are less constrained temporally, allowing the user more freedom with respect to their relative timing. Of course, that freedom is not necessarily exercised by the user at run-time; given that a set of actions *can be* interleaved by the user, it does not

Figure 16. Activity waveform diagram of sequenced tasks.

follow that they *are* interleaved. From the point of view of describing human-computer interaction for design purposes, the interesting relationships are those that express the *possibilities* for actions.

As mentioned at the end of Section 1, readers not wishing to get into the details of symbols and equations can skip the numbered equations in Section 9 and still understand most of the concepts. Each equation is preceded by a prose description.

9.1. Sequence

Perhaps the simplest temporal relationship between two tasks is that which is expressed by the binary relation *sequence*; one task is performed immediately and entirely after the other. More formally, two user actions, α_1 and α_2 , are in sequence (related by the sequence relation, S) if and only if the entire lifetime of α_1 immediately precedes the lifetime of α_2 :

$$\alpha_1 S \alpha_2 \leftrightarrow ((L(\alpha_1) P L(\alpha_2)) \wedge \neg \exists \pi_i(\alpha_j)((L(\alpha_1) P \pi_i(\alpha_j)) \wedge (\pi_i(\alpha_j) P L(\alpha_2))))), \text{ for } j \neq 1, j \neq 2 \quad (15)$$

Note that this sense of sequence, which does not allow an intervening action between two actions in sequence, can be thought of as a strong precedence relation. This observation will be of importance when examining the concepts of interleaving and interruption. Figure 16 is an activity waveform diagram illustrating a sequence. The actions shown here could all be different or could involve different instances of the same task. Notice that each action instance is performed to completion before another is begun; no interruption is occurring.

In the UAN, a sequence is represented in the following way. The S is dropped, and the temporal sequence of actions, α_1 and α_2 , is represented iconographically by writing the actions as a spatial sequence horizontally:

$$\alpha_1 \alpha_2$$

or vertically:

α_1
 α_2

As an example from the CMS, a high-level task may be defined as the sequence of two other tasks (as in Figure 9):

Task: add_appointment
access_appointment
edit_appointment

The task of adding an appointment is defined to be the sequence of accessing the appropriate appointment followed by the editing (including typing, corrections, etc. in predefined fields in the time slot) of the appointment.

So far, a sequence is a binary temporal relation; that is, it applies to exactly two tasks as operands. There are two ways that sequences, and the other temporal relations, can be applied on a larger scope. One way is to build up levels of abstraction; the second way is by grouping with parentheses. In the next two subsections, it is convenient to define these two methods of expansion in terms of sequences, but the concepts apply to each of the temporal relations equally well.

Task Names and Levels of Abstraction

A task description written in the UAN is a set of actions interspersed with temporal operators according to the rules for their application, following Definitions 7 and 8. This task can then be named, and the name is used as a reference to the task. This name reference is used as an action in another (higher level or containing) task. As an example, consider a simple task that has only a sequence, $\alpha_1 \alpha_2$. This task can be named " β ," and then β can be used in task γ in sequence with some other task ϵ . The use of a task name as a user action corresponds at run-time to the invocation of a user-performed procedure. The use of a task name as a reference to the task is an *invocation* and serves two purposes (just as invocations do in programming systems): abstraction (hiding the details of the procedure) and instantiation (creating a task instance—see Section 7.2—and giving it a lifetime).

The recursive nature of this abstraction operation makes it possible to build layers of abstraction, allowing the entire interface design to be organized into a quasi-hierarchical user task structure. Just as in the case of program code, the levels of abstraction are necessary for controlling complexity to promote understanding by readers and writers of the UAN. Also, just as in the case of software procedures and their calling structure, there will be a path of active

tasks down to the level of primitives. To illustrate with the example just given, consider the performance of task γ . At some time β will be invoked from within γ . During the performance of β , task α_1 will also be performed. At that moment, all of the tasks α_1 , β , and γ will be active. This kind of simultaneity is only an artifact of the hierarchical decomposition structure of tasks; a "calling" task and a "called" task will always have overlapping lifetimes. This is not the same, however, as two independent tasks being interleaved or concurrent. All the temporal relations described in this article are applied to independent tasks at the same level of abstraction—not between calling and called tasks in the task hierarchy.

Grouping, Closure, and Composition of Relations

An instance of a temporal relation between two tasks can be enclosed within parentheses. The effect is similar to the grouping into a named task as described in the previous section, except the resulting task is not named. For example, the sequence of actions:

$$\alpha_1 \alpha_2$$

can be grouped with parentheses into the following task:

$$(\alpha_1 \alpha_2).$$

Each of the temporal relations, R , maps a pair of actions into a task:

$$R: \{\text{actions}\} \times \{\text{actions}\} \rightarrow \{\text{tasks}\} \quad (16)$$

and a task is also an action; thus the temporal relations are closed over the set of all actions. Therefore, by composition, one can apply another relation between a group in parentheses and some third action, α_3 , yielding a new task, as in the case of this sequence:

$$(\alpha_1 \alpha_2) \alpha_3.$$

Composition of relations allows large task description to be built up of user actions (especially tasks) and temporal relations.

Applying the concept of grouping to the sequence relation, one can derive the property of associativity directly from the definition of the sequence relation in Equation 15:

$$(\alpha_1 \alpha_2) \alpha_3 = \alpha_1 (\alpha_2 \alpha_3) \quad (17)$$

The binary sequence relation can be generalized to the ternary case by extending Equation 17 in this way:

$$(\alpha_1 \alpha_2) \alpha_3 = \alpha_1 (\alpha_2 \alpha_3) = (\alpha_1 \alpha_2 \alpha_3) = \alpha_1 \alpha_2 \alpha_3 \quad (18)$$

Similarly, the sequence relation can be extended to the n-ary case:

$$\alpha_1 \alpha_2 \alpha_3 \dots \alpha_n \quad (19)$$

9.2. Waiting

Sometimes an interface designer wishes to constrain the time interval between tasks in a sequence. For example, to define a close relationship that combines two tasks into one, the interval could be required to be less than some time value. To illustrate, two mouse-button clicks, when performed within a short interval, are to be recognized as a distinct user action called a *double click*. In such cases where waiting is significant in a task description, the waiting interval acts as a temporal relation between the actions, constraining the temporal distance between actions in a sequence. Within a UAN task description, a *waiting* relation between tasks α_1 and α_2 is written as:

$$\alpha_1 (t \text{ comparison-operator } n) \alpha_2 \quad (20)$$

where t is the time to wait, comparison-operator makes an arithmetic comparison (such as less than or greater than), and n is a numeric value in units of time.

The example of the double click of a mouse button is represented in this specific UAN expression:

$$MVA (t < n) MVA$$

where MVA denotes the mouse button being depressed and released (clicked) and $(t < n)$ declares that the wait between mouse clicks must be less than n units of time. If the user waits longer than n time units, this action will be seen as two single mouse clicks. The value of n can be controlled by the user via an interface setting.

Another way waiting can be used in a UAN description as a temporal relation between two tasks is to indicate a minimum wait to cause some kind of time-out by the system:

$$\alpha_1 (t > n) \alpha_2$$

9.3. Repeating Disjunction

The vertical bar (|) is used to indicate a disjunction of choices among user tasks. For example, $\alpha_1 | \alpha_2 | \alpha_3$ denotes a three-way choice among α_1 , α_2 , and α_3 . A common high-level construct in the UAN is seen in this example of a *repeating disjunction*:

$$(\alpha_1 | \alpha_2 | \alpha_3)^*$$

This notation means that tasks α_1 , α_2 , and α_3 are initially equally available. The * means that the disjunction (the whole task within parentheses) is repeated any number of times. Once a task from the disjunction is begun, it is performed to completion, at which time the three tasks are equally available again. The cycle continues arbitrarily each time any one of the three tasks is selected by the user and performed to completion.

As an example from the CMS, the highest level task is defined in Figure 4 as the repeating disjunction of the five main user operations:

```
Task: manage_calendar
(access_appointment
| add_appointment
| update_appointment
| delete_appointment
| establish_alarm)*
```

Repeating disjunction is also used in the `access_appointment` task definition of Figure 5:

```
Task: access_appointment
(search
| access_month
| access_week
| access_day)*
access_time_slot
```

Observable user behavior in performing the `access_appointment` task is a series of instances from among the five tasks of `search`, `access_month`, `access_week`, and so on. Ways in which the user might decide which choices to make can be described within the `access_appointment` task using cognitive, perceptual, and decision-making activities, but these are not in the scope of the present article.

9.4. Order Independence



In the use of interactive computer systems, as in the world outside interfaces, it is not uncommon to find situations in which several tasks are to be performed but the order of their performance is immaterial. In the UAN, two user actions, α_1 and α_2 , are *order independent* if and only if both actions are required, but the lifetime of either may precede that of the other:



$$\alpha_1 \ \& \ \alpha_2 \leftrightarrow ((\alpha_1 \ \alpha_2) \mid (\alpha_2 \ \alpha_1)) \quad (21)$$

The order independence relation is not associative, but it nonetheless can be extended to the n-ary case:

$$\alpha_1 \ \& \ \alpha_2 \ \& \ . \ . \ . \ \& \ \alpha_n \quad (22)$$

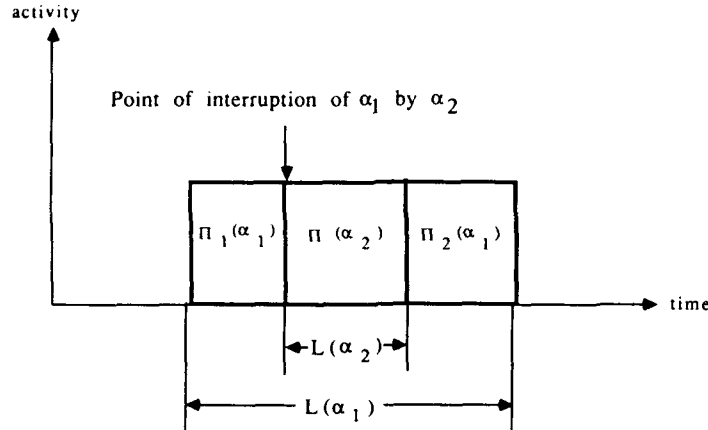
where this expression denotes a disjunction of all the possible sequential orderings of the actions. In practical terms, this means that all of the tasks— $\alpha_1, \alpha_2, \dots, \alpha_n$ —must be performed but that any order among them is acceptable.

An example of order independence at a very low user action level is seen in the task of entering a “command-X” on a Macintosh keyboard—a combination of the “” and “X” keys. The UAN uses “v” to denote the depressing of an input device such as a key or mouse button. The symbol v is used to indicate the release of such a device. The symbol on the key is the name of the device that is the key. Because the  key must be depressed before the X key, but the order of their release does not matter, the task is defined in the UAN as:

Task: command_X
 v Xv ( ^ & X^)

The `edit_appointment` task provides an example of order independence from the CMS. Suppose an appointment object has text fields for name of person, description of appointment, and location. The task of editing an appointment breaks down into the set of tasks for editing these smaller objects, and the order in which they are edited does not matter:

Task: edit_appointment
 view_level = time_slot:
 (edit_person
 & edit_description
 & edit_location)

Figure 17. The simplest case of interruption.

The `edit_person`, `edit_description`, and `edit_location` tasks will feature repeating disjunctions of editing subtasks such as `type_string`, `select_string`, `cut_string`, `copy_string`, `paste_string`, and the like.

9.5. Interruptibility

We begin by refining the concept of interruption, introduced earlier in Section 8.4. An instance of an action, α_2 , is interrupted by another action, α_1 , if and only if a period of activity of α_1 overlaps the lifetime of α_2 but does not overlap a period of activity of α_2 :

$$\alpha_2 \text{ is interrupted by } \alpha_1 \leftrightarrow \exists \pi_i(\alpha_1)((\pi_i(\alpha_1) \circ L(\alpha_2)) \wedge \neg \exists \pi_j(\alpha_2)(\pi_i(\alpha_1) \circ \pi_j(\alpha_2))) \quad (23)$$

The simplest case of interruption is shown in Figure 17. Task α_1 is begun and task α_2 interrupts, dividing α_1 into two periods of activity, $\pi_1(\alpha_1)$ and $\pi_2(\alpha_1)$. The lifetime of α_1 , $L(\alpha_1)$, spans the two periods of activity.

Because a design representation is intensional, there is no symbol in the UAN for "is interrupted by." Rather, there is a temporal operator to denote cases of interruptibility, cases where interruption can occur. Thus, the definition of interruptibility requires the use of alethic (truth-related) modalities in our expressions. That is, the defining proposition must assert the possibility of a certain state of affairs. For this purpose, we add to the first-order predicate calculus used so far the primitive monadic modal operator, M (Hughes & Cresswell, 1968), with the following definition:

$$Mp = \text{it is possible that } p \text{ (i.e., it is not a tautology that } \neg p) \quad (24)$$

Note that M expresses an alethic rather than a temporal (time-related) modality; although we are speaking of temporal relations, we do not use temporal modes.

An instance of an action, α_2 , is defined to be *interruptible* by another action, α_1 (α_1 can interrupt α_2), if and only if a period of activity of α_1 can overlap the lifetime of α_2 but cannot overlap a period of activity of α_2 :

$$\alpha_1 \rightarrow \alpha_2 \leftrightarrow M(\exists \pi_i(\alpha_1)((\pi_i(\alpha_1) \circ L(\alpha_2)) \wedge \neg \exists \pi_j(\alpha_2)(\pi_i(\alpha_1) \circ \pi_j(\alpha_2)))) \quad (25)$$

The interruptibility relation is not symmetric; $\alpha_1 \rightarrow \alpha_2$ implies neither $\alpha_2 \rightarrow \alpha_1$ nor $\neg(\alpha_2 \rightarrow \alpha_1)$.

Uninterruptible Tasks and Preemptive States

Consider a task α for which the action set is $A(\alpha)$. If task α' can interrupt task α , $\alpha' \rightarrow \alpha$, there are two ways that the definition in Equation 25 can be satisfied: α' can interrupt *between* the lifetimes of instances of the $\alpha_i \in A(\alpha)$, or interruption can occur *during* the lifetime of an α_i ; that is, $\alpha \rightarrow \alpha_i$. The general interpretation of the interruptibility relation includes both these cases.

It is also necessary to be able to define exceptions to this second case, namely, to be able to specify those α_i for which $\neg(\alpha' \rightarrow \alpha_i)$. One kind of exception occurs when α_i is primitive, denoted by the unary relation ($\text{prim}(\alpha)$). Primitive user actions are not interruptible.

A second situation in which a task instance must be specified as uninterruptible occurs in preemptive interface features (Thimbleby, 1990). A dialogue box is a good example. While using a dialogue box in task α_1 , a user generally cannot click in the window of task α_2 to change tasks until the dialogue box is exited. Preemptive states correspond to sets of user actions, the boundaries of which cannot be crossed by the interleaving relation. In other words, while in the dialogue box, the user can still interleave tasks but only among tasks within the dialogue box. In the UAN, pointed brackets, \langle, \rangle , represent the unary relation "is uninterruptible," enclosing those parts of a task description that are uninterruptible by other user actions at any level. For example, $\langle \alpha_1 \alpha_2 \alpha_3 \rangle$ denotes that the sequence of these user action instances cannot be interrupted.

Preemptive states in this view are a means of partitioning the user's task domain. A preemptive state is a task subdomain with circumscribed asynchronism. A preemptive state limits the user to a set of tasks usually disjoint from those available outside that state. Consider the graph or set of graphs that is the nondeterministic state transition diagram of the dialogue control for an interface. The part of the dialogue without preemptive states

can be considered the main dialogue. The main dialogue and the set of preemptive states would each be simply-connected components of the graph, the preemptive states being isolated from the rest of the interface except for the single transitions entering and leaving the preemptive state set. Modes are usually preemptive; consider the input mode in the Unix "vi" editor. There are many commands that lead to the input mode (open line, append, input, etc.) at which point almost all keystrokes are considered as input text. The Escape key allows the user to leave the input mode, and many vi keyboard commands once again become active. Inputs that apply in the input mode are more or less disjoint from the commands that apply to vi outside the input mode.

Modes and preemptive states in interface designs are the result of decisions (conscious or not) about the task domain. It is not our intention to argue for or against such decisions here, only to be able to represent the designs.

Scope of Interruptibility

To understand the effect of interruptibility on a task or action, α , it is useful to determine which subtasks (tasks or actions invoked by α) themselves are interruptible. We must begin by formalizing the concept of invocation, introduced in Sections 7.2 and 9.1. User action α' can directly invoke user action α (α is directly invocable by α') if and only if α is a member of the action set of α' :

$$\alpha' \underline{>>} \alpha \leftrightarrow \alpha \in A(\alpha') \quad (26)$$

Action α' can invoke action α (α is invocable by α') if and only if there is a progression of possible direct invocations, $\alpha_1, \alpha_2, \dots, \alpha_k$, connecting α' and α :

$$\begin{aligned} \alpha' >> \alpha &\leftrightarrow \exists (\alpha_1, \alpha_2, \dots, \alpha_k) \ni \\ &\text{i. } \alpha' \underline{>>} \alpha_1 \\ &\text{ii. } \alpha_i \underline{>>} \alpha_{i+1}, \text{ for } i = 1, 2, \dots, k - 1 \\ &\text{iii. } \alpha_k \underline{>>} \alpha \end{aligned} \quad (27)$$

It follows that $\alpha' \underline{>>} \alpha \supset \alpha' >> \alpha$, where $\alpha_1, \alpha_2, \dots, \alpha_k$ is a null progression.

In like manner, we define the "can interruptibly invoke" relation (\bullet). Action α' can interruptibly invoke action α (α is interruptibly invocable by α') if and only if α' can invoke α and α is neither uninterruptible nor a primitive.

$$\alpha' \bullet \alpha \leftrightarrow ((\alpha' >> \alpha) \wedge \neg (<\alpha> \text{ or } \text{prim}(\alpha))) \quad (28)$$

If $\alpha' \rightarrow \alpha$, the full set of user actions collectively known as the *scope of interruptibility* is α plus all the user actions invocable by α , except primitives

and uninterruptible actions; that is, the scope of interruptibility is α and all user actions interruptibly invocable by α :

$$I(\alpha', \alpha) = \{\alpha\} \cup \{\alpha'' \mid \alpha \cdot \alpha''\} \quad (29)$$

9.6. One-Way Interleavability

There may be times when the interface designer wishes to specify $\alpha_1 \rightarrow \alpha_2 \wedge \neg(\alpha_2 \rightarrow \alpha_1)$. For example, consider the case of help as a facility available during some other complex task such as the editing of a document. If the high-level task is described as follows:

$$\text{help} \rightarrow \text{edit document} \wedge \neg(\text{edit document} \rightarrow \text{help})$$

the user can invoke the help task at any time during the editing, but closure of the help task is required before editing can continue. In other words, help tasks can interrupt the editing, but editing cannot interrupt the help. We call this *one-way interleavability*.

An instance of an action, α_1 , is defined to be one-way interleavable with action α_2 if and only if α_1 can interrupt α_2 but α_2 cannot interrupt α_1 :

$$\alpha_1 \Rightarrow \alpha_2 \Leftrightarrow ((\alpha_1 \rightarrow \alpha_2) \wedge \neg(\alpha_2 \rightarrow \alpha_1)) \quad (30)$$

9.7. Mutual Interleavability

Two user actions, α_1 and α_2 , are *mutually interleavable* if and only if they can interrupt each other; that is, it is possible that a period of activity of either action can interrupt a period of activity of the other:

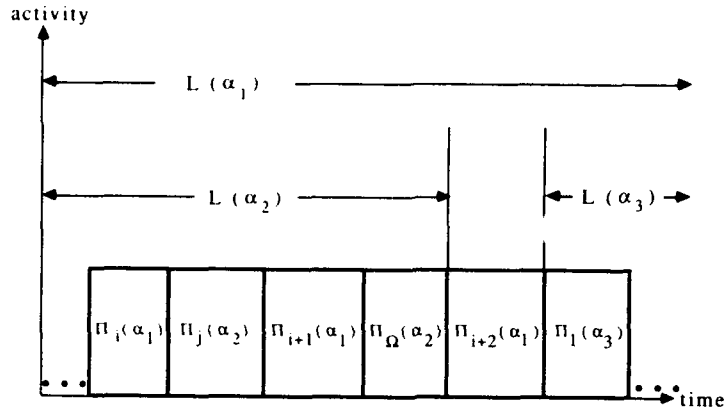
$$\alpha_1 \Leftrightarrow \alpha_2 \Leftrightarrow ((\alpha_1 \rightarrow \alpha_2) \wedge (\alpha_2 \rightarrow \alpha_1)) \quad (31)$$

Unqualified use of the terms *interleaving* and *interleavability* is reserved for the more general two-way (mutual) case. As previously discussed, $\alpha_1 \Leftrightarrow \alpha_2$ means that α_1 and α_2 are mutually interleavable throughout the scopes of interruptibility of α_1 and α_2 .

One can derive the following property of associativity directly from the definition of the interleavability relation in Equation 31:

$$(\alpha_1 \Leftrightarrow \alpha_2) \Leftrightarrow \alpha_3 = \alpha_1 \Leftrightarrow (\alpha_2 \Leftrightarrow \alpha_3) \quad (32)$$

The binary interleavability relation, again, can be generalized to the ternary case by extending Equation 32 in this way:

Figure 18. Activity waveform diagram of interleaved tasks.

$$(\alpha_1 \Leftrightarrow \alpha_2) \Leftrightarrow \alpha_3 = \alpha_1 \Leftrightarrow (\alpha_2 \Leftrightarrow \alpha_3) = (\alpha_1 \Leftrightarrow \alpha_2 \Leftrightarrow \alpha_3) = \alpha_1 \Leftrightarrow \alpha_2 \Leftrightarrow \alpha_3 \quad (33)$$

and to the n-ary case:

$$\alpha_1 \Leftrightarrow \alpha_2 \Leftrightarrow \dots \Leftrightarrow \alpha_n \quad (34)$$

Interleavability is one of the cases in which it is necessary to distinguish between tasks and instances of tasks (Section 7.2). At run-time it is the instances, of course, that are interleaved. Consider the case of help as a facility available during some other complex task (e.g., editing a document). Suppose that the help information, when invoked, appears in a separate window from the document being edited. The editing and help tasks are interleavable in that the user may alternate attention, and actions, from one window to the other. There is only one instance of each user task: one editing task and one help task. Additionally, it is possible that the user might terminate one help task during the editing task and subsequently start another interleaved help task while still within the initial editing session. This is a case of two instances of the same task type (e.g., help) being interleaved with a single instance of a distinct task (e.g., editing). Thus, $\alpha \Leftrightarrow \alpha$ does not mean that interleavability is reflexive; rather, this expression refers to the interleavability among different instances of α .

There are many possible configurations of interleaving. Figure 18 shows several interleaved tasks. Task α_1 is interleaved with α_2 and with α_3 , but α_2 is not interleaved with α_3 . Task α_2 is not interrupted by α_1 after the period of activity π_Ω , because this period is the termination of α_2 . In general α_1 , α_2 , and α_3 are different tasks, but again it is possible for interleaving to involve

different instances of the same task. For example, two help windows could be open simultaneously, with the user shifting attention from editing to each of the help windows alternately.

An example from the CMS can be used to illustrate interleaving. The five main user operations shown in Figure 4 are subtasks of the main task, **manage_calendar**. In Sections 4 and 9.3, these were represented as a repeating choice. A more asynchronous design would allow an instance of each subtask to be created in its own window. The user could go back and forth, interleaving activity among the subtasks by activating one window after another (e.g., by clicking in each window). The task description for this interleaved design is:

Task: manage_calendar
 (access_appointment
 \Leftrightarrow add_appointment
 \Leftrightarrow update_appointment
 \Leftrightarrow delete_appointment
 \Leftrightarrow establish_alarm)*

9.8. Concurrency

Two user actions, α_1 and α_2 , can be concurrent if and only if it is possible that a period of activity of one can overlap a period of activity of the other:

$$\alpha_1 \parallel \alpha_2 \leftrightarrow M(\exists \pi_i(\alpha_1) \exists \pi_j(\alpha_2) (\pi_i(\alpha_1) \circ \pi_j(\alpha_2))) \quad (35)$$

where M is the modal operator defined in Equation 24. In Figure 19, tasks α_1 and α_2 are concurrent. A period of activity in α_1 is overlapped by periods of activity of α_2 .

One can derive the following property of associativity directly from the definition of the concurrency relation in Equation 35:

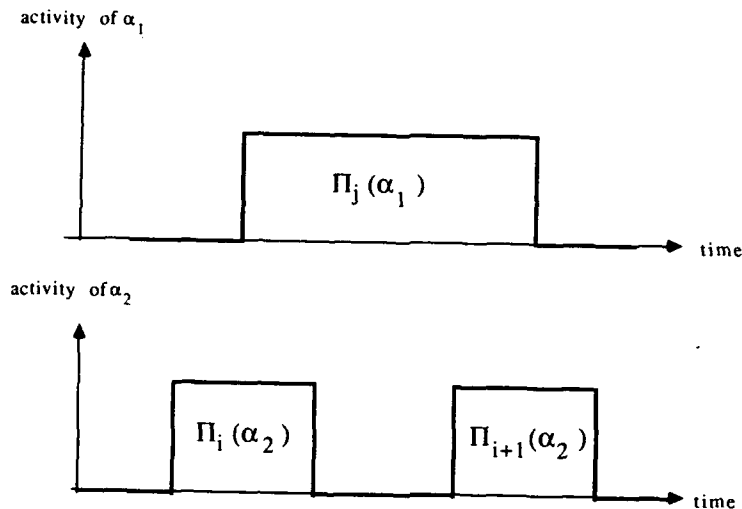
$$(\alpha_1 \parallel \alpha_2) \parallel \alpha_3 = \alpha_1 \parallel (\alpha_2 \parallel \alpha_3) \quad (36)$$

This can be generalized to the ternary case:

$$(\alpha_1 \parallel \alpha_2) \parallel \alpha_3 = \alpha_1 \parallel (\alpha_2 \parallel \alpha_3) = (\alpha_1 \parallel \alpha_2 \parallel \alpha_3) = \alpha_1 \parallel \alpha_2 \parallel \alpha_3 \quad (37)$$

and to the n-ary case:

$$\alpha_1 \parallel \alpha_2 \parallel \dots \parallel \alpha_n \quad (38)$$

Figure 19. Activity waveform diagram of concurrent tasks.

Concurrency is a temporal relation that has not been greatly exploited in user interfaces. This may be because users are not skilled enough to carry out tasks concurrently. Norman (1988) noted that much conscious activity is both relatively slow and sequential in nature. We are able to switch attention from one task to another, or even transfer information to and from tasks. But this is interleaving, not concurrency, of action. Nevertheless, there are cases in which it is possible and, indeed, preferable, to carry out more than one task at the same time. A user can be perceptually responsive to information on the display while typing or manipulating the mouse. Buxton (1983) described input techniques that rely on the use of both hands concurrently. Such situations require the full power of the concurrency relation as described before.

Another kind of concurrency is seen in the actions of two or more users doing computer-supported cooperative work. These users, using different workstations, may be able to perform actions simultaneously on shared instances of application objects, possibly operating through different views. For a representation of the CMS in the case where periods of activity among the tasks can overlap, the task description becomes:

Task: manage_calendar
 (access_appointment
 | add_appointment
 | update_appointment
 | delete_appointment
 | establish_alarm)*

10. DISCUSSION

10.1. How the UAN Helps With Interface Development

User interface design has two separate parts: design of the user interaction and design of the corresponding user interface software. The interaction designer receives, as inputs, requirements for the design from the systems analysis process, which in turn includes inputs from marketing, task analysis, user analysis, needs analysis, and so forth. The interaction designer—who works in the behavioral domain of tasks, user actions, and perceived feedback—produces as output a behavioral design of the user interaction part of the interface. This interaction design now becomes the requirements for the user interface software designers and implementers. A precise and formal technique is needed to convey these requirements independently of the software by which the interaction is implemented. In conjunction with screen pictures showing interface objects and state diagrams showing user modes and interface states, the UAN is such a technique. Because no behavioral representation technique appropriate for documenting interaction design previously existed, current practice has been to use software objects (e.g., widgets from software toolkits) directly for interaction design representation.

At this point in the interface development process, the design is often set in a prototype, the beginning of a commitment to a software embodiment. Although the prototype is used for some kinds of formative evaluation (e.g., user testing), a behavioral representation of the design offers advantages for other kinds of evaluation. One is analytical evaluation, analysis (probably automated) of the design in search of inconsistency, ambiguity, and other undesirable characteristics. This kind of analysis is possible with the UAN, but so far it is still in the category of future work. Another important kind of evaluation at this point in the development process is a design walk-through, which typically involves designers, evaluators, and possibly implementers. Our experience has been that the UAN design representation is typically more accurate, more complete, and more precise than the prototype as a source for answers to the questions that arise in a design walk-through (e.g., about how a particular interface feature or task actually works). Also, because a prototype yields information about the design by showing examples of its operation, it is extensional or instance oriented. In contrast, a UAN representation is intensional and, therefore, explicitly states all possibilities. For example, consider a simple task that is the disjunction of tasks A and B (task: $A \mid B$). The UAN notation makes it immediately evident that there are precisely two alternatives from which to choose. Using the prototype, one might try task A and see that it works, but then possibly not realize task B had also been possible at that time (especially if the operation of task A in the

prototype makes task B subsequently unavailable). In addition, the prototype does not generally convey whether there is some other task C also available.

Although this intensional capability of the UAN is important for analytic evaluation and design walk-throughs, it is essential for conveying the behavioral design of the interaction to user interface software designers and implementers. Here, an extensional prototype simply does not suffice; solid and precise intensional specifications are a necessity.

10.2. Conclusions

As one moves from one temporal relation to another, from sequence to order independent, interleavable, and concurrent, it is in a direction of decreasing temporal constraints. The temporal nature of a sequence is quite constrained. The first action must be performed completely, then the next, and so on, until all the actions are completed. In many sequential interface designs, this constraint is arbitrary and even opposed to the cognitive and task needs of the user. For example, the initiation of a second task in the middle of a first task may be very useful in order to get information necessary for the completion of the first task. In this case, an interface design to support the user would allow the second task to be interleaved, so it does not destroy the context of the first task.

The repeating disjunction allows a very limited measure of asynchronism at a high level by allowing a choice of tasks. Once a sequence is initiated, however, no interruption, interleaving, or concurrency is allowed.

With order independence, all actions must be performed and each one completed before another is begun. But the constraint on the specific ordering among the actions is removed. Interleaving removes the constraint of being performed to completion before beginning another action, allowing an action to be interrupted.

Interleavability and concurrency are defined in different ways, but they share the fact that lifetimes of tasks can overlap. But interleavability means periods of activity cannot overlap, whereas they can in concurrency. The difference between interleaving and concurrency is something like the difference between real and apparent concurrency in an operating system (Lorin, 1972). Although multiprogramming is based on interleaving of processes, for many purposes the processes are regarded as being concurrent. Real concurrency, however, requires multiprocessing, such as is found between a central processor and an input/output processor (channel) in the hardware. Similarly, one can see the difference between interleaving and concurrency at the level of physical user actions, but perhaps not always at the level of the user's mental model of the tasks.

Analysis of the various cases using temporal relations gives the designer the

ability to distinguish task types that are significantly different but that, without these relations, would be difficult to identify. Furthermore, adding operators to the UAN to express these relations gives the designer a powerful means of representing such interfaces.

Acknowledgments. The authors acknowledge Dr. Antonio Siochi as the originator of the UAN. We also gratefully acknowledge helpful discussions with Dr. Deborah Hix, Dr. Kevin Waite, Cathy Wood, Dr. Stephen Draper, and Dr. John Urquhart. We thank the anonymous reviewers and the associate editor, Dr. Ruven Brooks, for careful thought and detailed suggestions that greatly improved the article. Dr. Marilyn Mantei originally created the CMS as an example exercise in interface design. Much mileage has been gotten from it since.

Support. The UAN was created in the Virginia Tech Dialogue Management Project during work sponsored by the Software Productivity Consortium and the Virginia Center for Innovative Technology. Partial support for our study was received by Grant Number IRI-9023333 from the National Science Foundation under the supervision of Dr. John Hestenes.

REFERENCES

- Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 11, 832-843.
- Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23, 123-154.
- Barringer, H. A. (1985). *Survey of verification techniques for parallel programs, lecture notes in computer science No. 191*. Berlin: Springer-Verlag.
- Buxton, W. (1983). Lexical and pragmatic considerations of input structures. *Computer Graphics*, 17, 31-37.
- Card, S. K., & Moran, T. P. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23, 396-410.
- Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Cardelli, L., & Pike, R. (1985). Squeak: A language for communicating with mice. *Computer Graphics*, 19, 199-204.
- Decortis, F., & De Keyser, V. (1988). Time: The Cinderella of man-machine interaction. *Proceedings of IFAC Man-Machine Systems*, pp. 123-128.
- Duce, D. A. (1985). Concerning the specification of user interfaces. *Computer Graphics Forum*, 4, 251-258.
- Green, M. (1985). The University of Alberta user interface management system. *Computer Graphics*, 19, 205-213.
- Green, M. (1986). A survey of three dialog models. *ACM Transactions on Graphics*, 5, 244-275.
- Hale, R. (1987). Temporal logic programming. In A. Galton (Ed.), *Temporal logics and their applications* (pp. 91-119). London: Academic.
- Hartson, H. R., & Hix, D. (1989). Toward empirically derived methodologies and

- tools for human-computer interface development. *International Journal of Man-Machine Studies*, 31, 477-494.
- Hartson, H. R., Siochi, A. C., & Hix, D. (1990). The UAN: A user-oriented representation for direct manipulation interface designs. *ACM Transactions on Information Systems*, 8, 181-203.
- Hawking, S. W. (1988). *A brief history of time*. Toronto: Bantam.
- Hill, R. (1987). Event-response systems—A technique for specifying multi-threaded dialogues. *Proceedings of the CHI+GI '87 Conference on Human Factors in Computing Systems*, 241-248. New York: ACM.
- Hill, R., & Hermann, M. (1989). The structure of Tube: A tool for constructing advanced user interfaces. *Proceedings of Eurographics '89*, pp. 15-25.
- Hughes, G. E., & Cresswell, M. J. (1968). *An introduction to modal logic*. London: Methuen.
- Jacob, R. J. K. (1986). A specification language for direct manipulation user interfaces. *ACM Transactions on Graphics*, 5, 283-317.
- Kahn, K., & Gorry, G. A. (1977). Mechanizing temporal knowledge. *Artificial Intelligence*, 9, 87-108.
- Kieras, D., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22, 365-394.
- Kowalski, R. A., & Sergot, M. J. (1986). A logic-based calculus of events. *New Generation Computing*, 4, 67-95.
- Lorin, H. (1972). *Parallelism in hardware and software: Real and apparent concurrency*. Englewood Cliffs, NJ: Prentice-Hall.
- McDermott, D. A. (1982). Temporal logic for reasoning about processes and plans. *Cognitive Science*, 6, 101-155.
- Moran, T. P. (1981). The command language grammar. A representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies*, 15, 3-51.
- Moszkowski, B. C. (1986). *Executing temporal logic programs*. Cambridge, England: Cambridge University Press.
- Myers, B. (1987). Creating dynamic interaction techniques by demonstration. *Proceedings of the CHI+GI '87 Conference on Human Factors in Computing Systems*, 271-278. New York: ACM.
- Nickerson, R. S., & Pew, R. W. (1990). User-friendlier interface. *IEEE Spectrum*, 27(7), 40-43.
- Norman, D. A. (1988). *The psychology of everyday things*. New York: Basic Books.
- Norman, D. A., & Draper, S. W. (1986). *User centered system design: New perspectives on human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Olsen, D. R., Jr., & Dempsey, E. P. (1983). Syngraph: A graphical user interface generator. *Computer Graphics*, 17, 43-50.
- Payne, S. J., & Green, T. R. G. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction*, 2, 93-133.
- Reisner, P. (1981). Formal grammar and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering*, SE-7, 229-240.
- Richards, J. T., Boies, S. J., & Gould, J. D. (1986). Rapid prototyping and system development: Examination of an interface toolkit for voice and telephony applications. *Proceedings of the CHI '86 Conference on Human Factors in Computing Systems*, 216-220. New York: ACM.

- Sharratt, B. (1990). Memory-cognition-action tables: A pragmatic approach to analytical modelling. *Proceedings of Interact '90*, 271-275. Amsterdam: Elsevier.
- Shneiderman, B. (1982). Multi-party grammars and related features for designing interactive systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 12, 148-154.
- Siochi, A. C., & Hartson, H. R. (1989). Task-oriented representation of asynchronous user interfaces. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems*, 183-188. New York: ACM.
- Thimbleby, H. (1990). *User interface design*. New York: ACM/Addison-Wesley.
- Wellner, P. D. (1989). Statemaster: A UIMS based on statecharts for prototyping and target implementation. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems*, 177-182. New York: ACM.

HCI Editorial Record. First manuscript received March 1, 1990. Revisions received December 7, 1990, and May 31, 1991. Accepted by Ruven Brooks. Final manuscript received September 17, 1991. — *Editor*

APPENDIX. MATHEMATICAL SYMBOLOGY

Symbol	Meaning
$\alpha R \beta$	α is related to β by relation R
\Leftrightarrow	If and only if
\ni	Such that
$ $	Such that (in set notation)
$ $	Disjunction or logical OR
\wedge	Logical AND
\neg	Logical NOT
\supset	Implies
\forall	For all
\exists	There exists a
\in	Is a member of (set)
\subset	Is a subset of
\times	Cartesian product (of sets)
\rightarrow	Maps to
\perp	Is a tautology (theorem) that
\cup	Set union
$\text{prim}(\alpha)$	Boolean, true if α is primitive
$A(\alpha)$	The action set of task α
$>>$	Can invoke
Mp	It is possible that p (is true)
$<\alpha>$	Task α is not interruptible
\bullet	Can interruptibly invoke

Note. The symbols are listed here approximately in the order of their appearance.

Inferring Graphical Procedures: The Compleat Metamouse

**David L. Maulsby, Ian H. Witten,
Kenneth A. Kittlitz, and Valerio G. Franceschin**
University of Calgary

ABSTRACT

Metamouse is a demonstrational interface for graphical editing tasks within a drawing program. The user specifies a procedure by performing an example execution trace and creating graphical tools where necessary to help make constraints explicit. The system generalizes the user's action sequence, identifying key features of individual steps and disregarding coincidental events. It creates a program with loops and conditional branches as appropriate and predicts upcoming actions, thereby reducing the tedium of repetitive and precise graphical editing. It uses default reasoning about graphical constraints to make initial generalizations and enables the user to correct these hypotheses either by rejecting its predictions or by editing iconic descriptors it displays after each action.

Authors' present addresses: David L. Maulsby, Kenneth A. Kittlitz, and Valerio G. Franceschin, Knowledge Sciences Laboratory, Department of Computer Science, University of Calgary, 2500 University Drive NW, Calgary, Alberta T2N 1N4, Canada; Ian H. Witten, Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand.

CONTENTS

1. INTRODUCTION
 2. APPLICATIONS
 - 2.1. Moving a Stove
 - 2.2. Sorting a Bar Chart
 - 2.3. Aligning Boxes
 - 2.4. Adapting "Align Boxes"
 3. BACKGROUND
 - 3.1. The Declarative Approach
 - 3.2. The Procedural Approach
 - 3.3. The Metamouse Approach
 - 3.4. Machine Learning and Generalization
 4. SYSTEM COMPONENTS
 - 4.1. Drawing Program
 - 4.2. Basil and the User Interface
 - 4.3. Overview of Learning Module
 - 4.4. Action Recorder
 - 4.5. Action Matcher
 - 4.6. Variable Inducer
 - 4.7. Constraint Classifier
 - 4.8. Constraint Solver
 5. EVALUATION
 - 5.1. Performance of Tasks
 - 5.2. Evaluating Interaction
 - Pilot Study
 - Controlled Study
 6. FUTURE WORK
 7. CONCLUSIONS
-

1. INTRODUCTION

The direct manipulation interface introduced in the Xerox Star (Johnson et al., 1989; Smith, Irby, Kimball, Verplank, & Harslem, 1982) and popularized by the Macintosh (Williams, 1984) has encouraged people to use computers in their writing, drawing, and management activities. A serious shortcoming of current interactive point-and-click systems is their failure to supply a natural way for end users to create programs within the user interface. Without programming, only those operations designed into the system are automated—The rest are left for the user to perform manually. Enriching the system's repertoire with libraries of special commands, and enriching commands with optional arguments (as in Unix), tends to alienate the very people who are drawn to the simplicity of direct manipulation. An alternative approach is to base a system on relatively few primitive operations

but to make it easy for end users to customize, creating their own procedures and importing those of others when desired.

End user programming conflicts with the idea of direct manipulation because programs must include abstractions of objects and relations—And direct manipulation is about concrete, literal communication between user and machine. Programming a sequence of menu selections by demonstration is simple; the difficulty comes with the need to use abstractions (like variables and functions) and control structures (like iteration and conditional branching) that are normally implicit within a task. Although they can be specified by annotating the demonstration (e.g., Halbert, 1984; Pence & Wakefield, 1988), this distracts from real work, tends to deter users from setting up programs, and (except in very simple tasks) is unsuited to those who have not been exposed to the art of programming. The alternative is to infer abstractions from the concrete traces that users provide. This will not be feasible unless search is restricted by focusing the system's attention on a small number of features at each step (Heise, 1989), and the system cannot reliably select these features itself without some help from the user and from domain knowledge.

Metamouse is a system for programming by example that helps users with annotation and focus of attention through a "coaching" metaphor. Users imagine they are training a graphical turtle named Basil. To work effectively, they must understand the limits on Basil's powers of perception and inference and be aware of his focus of attention. The system communicates this information economically by moving Basil to locations selected by the mouse, by highlighting objects he senses, and by asking questions through dialogue boxes (MacDonald & Witten, 1987). Throughout this article, *Basil* refers to the agent perceived by the user, and *Metamouse* refers to the underlying system.

Interaction supports the coaching metaphor in three ways. First, the Basil persona rationalizes the system's task model, including its focus of attention (nearby touch relations) and limits on its ability to make generalizations. Users understand that because Basil works by touch, measurements normally done "by eye" must be expressed by graphical construction. This extra information limits the search for generalizations but can be specified without abandoning the drawing program's direct-manipulation interface. Second, Basil demonstrates what he has learned at the earliest opportunity, so that users can benefit from it or correct it, as appropriate. Basil observes the user at work until he recognizes a pattern already learned, then predicts future actions, performing them for the user's approval. If he errs, or cannot find an action that fits the current situation, the user must resume demonstration. Third, Basil reacts immediately to the user's actions by providing feedback about postconditions, from which program variables and conditional opera-

tors are abstracted. Feedback is graphical and limited to a simple classification of Basil's perceptions—Objects and relations between them are highlighted one way if they are considered important and another if merely observed. Should Basil need more information about the current postconditions, he requests it through a pop-up dialogue that provides the possible replies.

In summary, Metamouse is an instructible system for graphical editing tasks. It learns complex, customized, iterative procedures based on a few editing primitives and serves as an easy, effective technique for programming. It learns incrementally, so that a procedure invoked under circumstances that differ from those in which it was taught may be extended (or generalized, or debugged) quickly and easily. This greatly increases the reusability of end user programs.

A prototype has been implemented that exhibits the basic structure and capabilities of an apprenticeship learning system. It observes the user's actions, performs a localized analysis of changes in spatial relations to isolate constraints, and matches action sequences to build a state graph that may contain conditional branches and loops. It induces variables for objects and distinguishes constants from run-time input parameters. It includes a symbolic (name-binding) and numeric (range-intersecting) constraint solver to perform the actions it has learned.

This article describes the system's design, implementation, and initial evaluation. We begin by illustrating how Metamouse can help the user with three particular graphical tasks. We review the history of tools to help automate graphical editing and contrast our approach with two main competitors, constraint-based and procedure-based systems, showing how it combines elements of both. We also briefly review similarity- and explanation-based generalization, techniques of machine learning that bear directly on demonstrational interfaces. Section 4 describes the components of the Metamouse system and how they work together. Section 5 evaluates its performance, first by showing how it copes with the three example tasks and then by describing a small experiment on how human subjects come to understand it. Finally, we appraise the limitations of the existing implementation and discuss how they might be overcome.

2. APPLICATIONS

Aesthetically pleasing, visually coherent, meaningful pictures are characterized by the spatial relations that group components, suggest relative importance, lead the eye through a visual narrative, and reveal subtle connections. With or without the help of a computer, a graphic artist must manage complex, competing relationships that may require compromise or careful ordering to be resolved. A formal computational model of such a design process is constraint resolution. Under this model, a drawing evolves

as objects and constraints are added, altered, or removed. The interaction of constraints is resolved (if possible) by update procedures; for instance, globally changing a typeface in a flowchart triggers an update to enlarge boxes, which triggers an update to reposition them. Performing such updates manually is repetitive work that demands precision, planning, and patience.

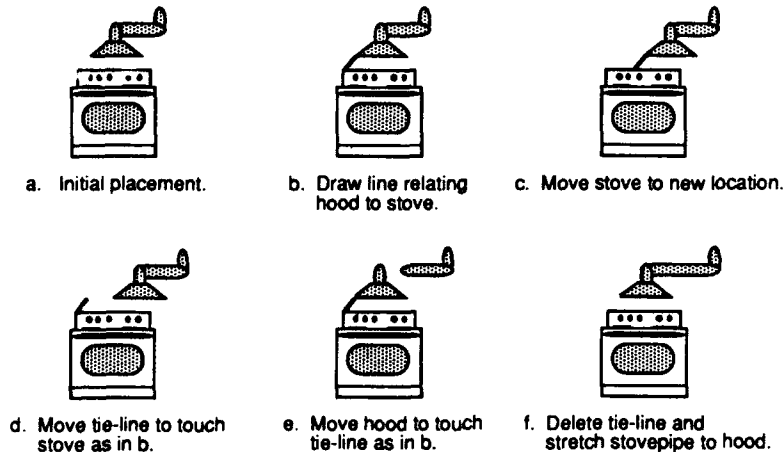
Metamouse automates a constraint update procedure by observing a sequence of edits demonstrated by the user, from which it infers action goals, variables, iteration, and branching. A goal is a conjunction of constraints to be achieved by a single editing step; for instance, if the user drags a box so that the midpoint of its top edge touches the end of one line and the midpoint of its right edge touches the end of another line, the action's goal is this pair of touch constraints. An action goal is a subgoal of the whole procedure.

Note that the term *constraint* in this article means "a spatial relation of special interest." In much of the user interface literature, exemplified by Sutherland (1963) and by Borning (1986), the term is restricted to relations that must persist as a drawing is altered and implies that constraint violations trigger update procedures. Our more general usage applies also to tasks that introduce and alter relations. Metamouse does not explicitly represent the goal of an entire procedure (which may be arbitrarily complex), so it does not trigger updates automatically. A feasible extension to the system would allow the user to attach a procedure to some editing action that affects a given set of objects, so that constraints (in the traditional sense) would be restored.

In the remainder of this section, we describe three tasks that exemplify important problems for users of interactive drafting packages: maintaining integrity of constraints throughout the editing process, coping with the tedium of repetition, and assimilating minor variations of a procedure. The need to achieve precision exceeding that of hand and eye, within a direct-manipulation system that shuns abstraction, underlies them all. The performance of the Metamouse system on these tasks is evaluated in Section 5.

2.1. Moving a Stove

The first task illustrates a procedure to maintain constraints when a picture is edited, the use of an auxiliary object (a tie-line) to visualize a primary constraint, and sequential demonstration to express constraint dependencies. Figure 1 shows what happens when a kitchen design is altered by moving the stove (note that graphics are less detailed in our drawing program, but the actions are identical). Whenever the designer moves the stove, the computer should respond by relocating the hood above the burners and stretching or shortening the stovepipe from the wall exit (cf. Figures 1a and 1f). The designer expresses the desired relative position of stove and hood by drawing a tie-line between them (Figure 1b). He or she demonstrates the procedure's

Figure 1. Maintaining constraints among objects.

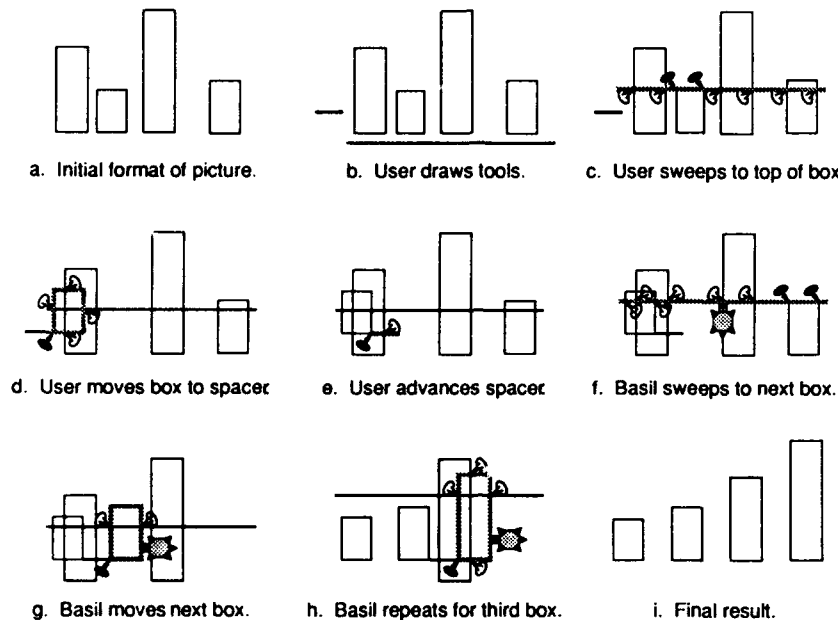
input and desired response by performing the complete edit as follows. First, he or she moves the stove (Figure 1c). The only change in touch (between tie-line and stove) is judged not to constrain the new position; by default this is an input. The user then drags the tie-line to touch the stove as before (Figure 1d). In the next steps, the user drags the hood to the tie-line (Figure 1e), deletes the tie-line, and stretches the stovepipe to the hood (Figure 1f). The complete procedure is named, stored, and added to a menu.

When reinvoked, the routine draws the tie-line between stove and hood, then asks the user to move the stove. When the user signals that he or she is done, the computer repositions the hood and reconnects the stovepipe, using the touch constraints it had inferred. If the user dragged the stove (say, downward) so that some constraint could not be solved (in this case, stretching the pipe to the hood), the system would ask him or her to demonstrate actions appropriate to that case.

2.2. Sorting a Bar Chart

In Figure 2, a set of rectangles is sorted by height and spaced in even intervals. This task illustrates iteration, precision, a selection rule, inputs, and constants. To teach it, the user must express implicit relationships such as distance and relative height using construction tools. Figure 2 shows the demonstration. Two display options have been set: show black tacks to indicate important touch relations, and show the turtle icon when predicting actions.

As a first step, the user draws a sweepline below the four boxes (Figure 2b). Finding no tactile constraints on its endpoints, Basil suggests they are inputs;

Figure 2. A group of boxes is sorted by height.

the user replies that they are constants. Next, the user draws a spacer at the left of the screen to control the distance between boxes (Figure 2b); it is an input. He or she picks the sweepline and drags it upward until it touches the top of a box—This selects the shortest one above it (Figure 2c). He or she moves that box to the near end of the spacer (Figure 2d), then relocates the spacer to its opposite side (Figure 2e). Although both box and spacer are touching other objects as well, Basil places a tack where they meet, to let the user know he considers this to be the only important constraint on these actions. When the user picks the sweepline a second time, Basil predicts the loop by moving the line upward until it touches the top of some box it has not touched in this way before (Figure 2f). Because the user accepts this loop, Basil performs all subsequent editing (Figures 2g–2h) until no box remains in the sweepline's path. Failure to find a box is the loop's terminating condition. Basil then asks the user to demonstrate the rest of the program, which involves removing the construction tools (Figure 2i) and signaling that the lesson is over.

When this program is later invoked from the tasks menu, Basil creates a sweepline at the same window coordinates as before. When he creates a spacer, he invites the user to edit it, because its position and length are inputs. Basil then performs the entire sort, regardless of the number of boxes.

2.3. Aligning Boxes

Figure 3 shows a set of boxes moved horizontally to an arbitrary guideline given by the user. As in the sorting task, constraint amongst a set of objects is implemented by positioning each one using an auxiliary "tool," in this case the guideline. In Figure 3, the display options are to show Basil at all times and to show not only black tacks but also white ones, which indicate incidental touches. The figure also includes the dialogue boxes through which Basil confirms hypotheses.

When the user draws the guideline (Figure 3a), its endpoints are unconstrained; the default selection in the dialogue box indicates that Basil assumes they are inputs. The user then draws a swepline (to ensure that boxes move horizontally); the fact that it crosses the guideline does not constrain its endpoints, so Basil assumes they are inputs also, but the user replies that the locations are constant (Figure 3b). The user picks the swepline (Figure 3c) and drags it up to the first box; Basil infers that contact with the bottom of the next higher box terminates this action and marks the constraints with black tacks (Figure 3d). The user grasps the same box and drags it to the point where the guideline and swepline cross; Basil infers that this three-way contact specifies where the box should go (Figure 3e). A third touch relation, between the box's lower left corner and the swepline, is marked with a white tack, indicating that Basil does not consider it a constraint. When the user reselects the swepline (Figure 3f), Basil conjectures a loop and confirms it by performing the remaining iterations (Figures 3g-3j). The first prediction involves searching for a box; Basil confirms that it should be somewhere above his current position (the "heading" dialogue in Figure 3g). At each step, Basil asks for confirmation. Loop termination is detected as in the sorting task (Figure 3k). The user completes the task by deleting the tools (Figure 3l).

2.4. Adapting "Align Boxes"

One of the advantages of using a learning system is that a new task may be taught as a variant of something the system already knows how to do. This increases the reusability of procedures: The user can invoke one that roughly fits the current problem, obtain immediate performance from those parts of it that are applicable, and manually perform (i.e., teach) the rest. An example of this is shown in Figure 4. This task differs from "align boxes" in several ways (see Figures 4a-4b): The box at the far left is to remain where it is; tie-lines into boxes' left sides must be reconnected, and tie-lines from the upper and lower box into the middle one are to be reconnected to the latter at

Figure 3. Teaching Basil to align a set of boxes.

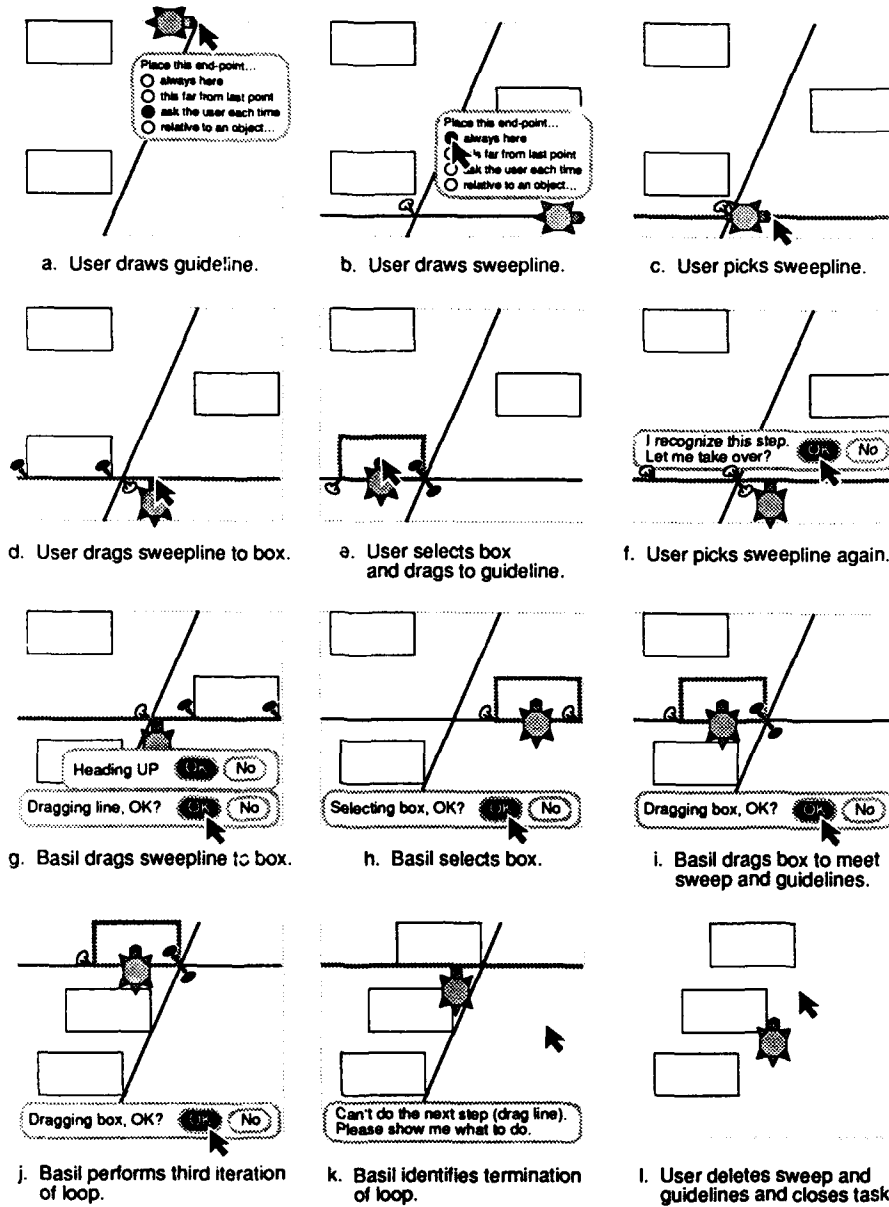
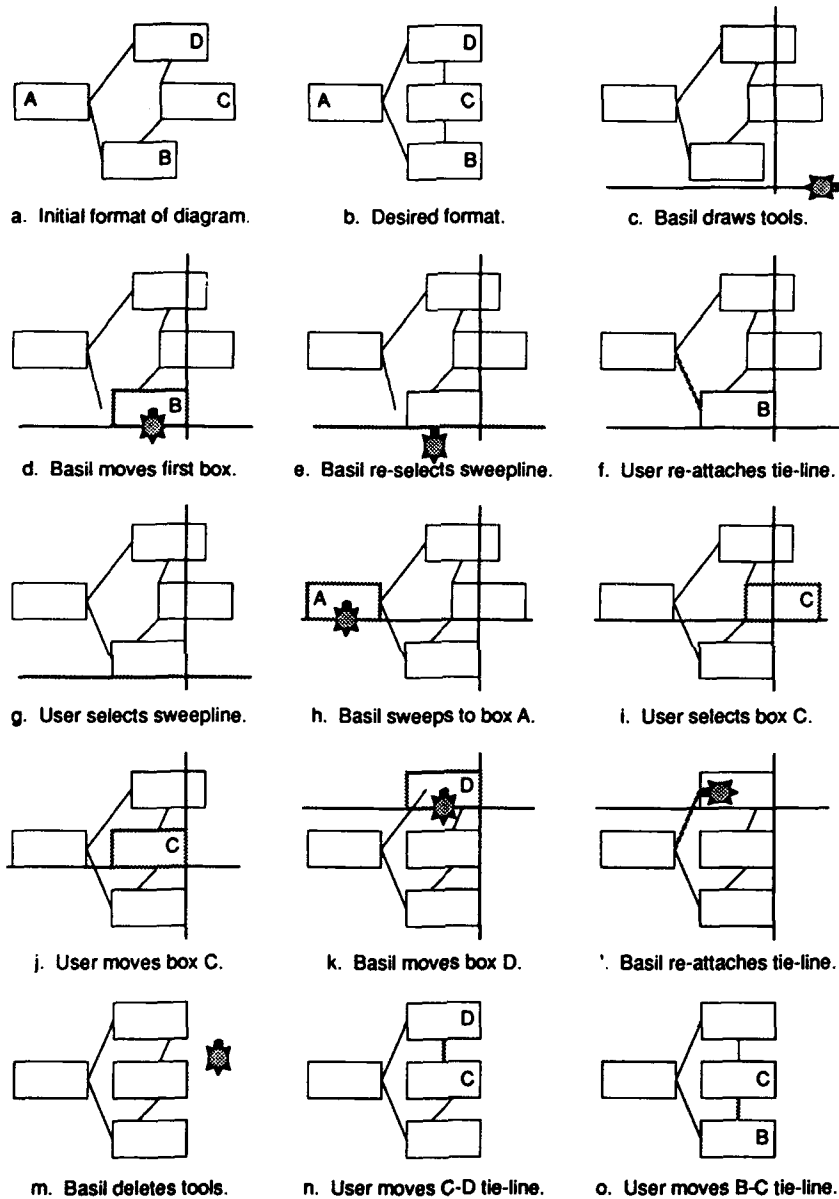
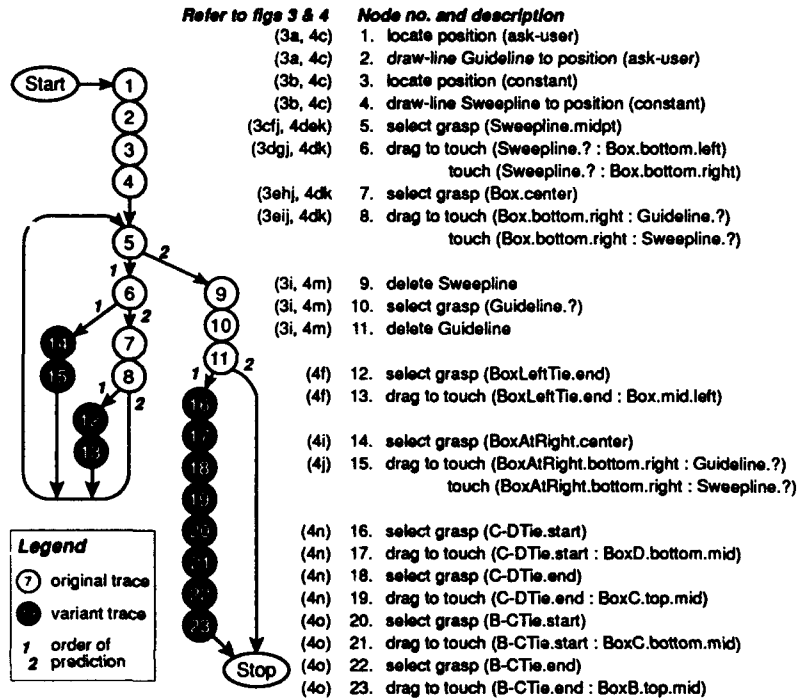


Figure 4. Aligning boxes and editing tie-lines.



the middle rather than the left edge. The resulting program is shown in Figure 5; white circles are nodes created for align boxes, and black ones are for the variant.

The user begins by invoking the align boxes task. Basil draws the guideline at its default position and invites the user to edit it. When he or she is done,

Figure 5. Program induced for "aligning boxes" task and variant.

Basil draws the sweepline (Figure 4c), drags it up to the first box, and performs the alignment (Figure 4d), all of which the user accepts. When Basil goes to grasp the sweepline again, the user stops him and reattaches the tie-line (Figure 4f). This creates a branch in which the new action will be predicted at higher priority (see Figure 5, transition from Nodes 8 to 12 vs. 5).

When the user picks the sweepline (Figure 4g), Basil recognizes this and conjectures a return to the main loop by dragging it up to the next box. Basil arbitrarily picks the one at the left (marked "A" in Figure 4h); the user rejects this and picks Box C (Figure 4i). This introduces another branch from Nodes 6 to 14 versus 7 (Figure 5). When the user moves C to the guideline (Figure 4j), Basil fails to match this with the existing program (due to our implementation's simplistic conventions on merging variables). The user's return to the sweepline is recognized, whereupon Basil drags it to Box D. He tries the new branch to Node 14, but it fails for lack of a second box at the sweepline. Instead, he follows the old branch, aligning Box D (Figure 4k) and, taking the branch to Node 12, reattaching its tie-line (Figure 4l).

Basil correctly exits from the loop and removes the tools (Figure 4m). When he predicts the end of the task, the user disagrees and edits the tie-lines from D to C (Figure 4n) and from C to B (Figure 4o), thus introducing a final

branch. The user can save the altered procedure as a new task or replace the old version. In either case, when it is reinvoked, the most recently taught branches are given priority, but their older alternatives are still accessible in case an entry condition fails.

This example illustrates the trading of control that can occur when debugging or adapting a procedure with Basil. These interruptions are worthwhile if the computer performs most of the work, or the most difficult parts of it, or if the procedure is to be reused often. Of 32 steps in this task, Basil performed 18. Improvements to Basil's generalization of actions would increase this to 20 (and eliminate the branch to Nodes 14 and 15). Of nine steps that involved precision positioning (aligning, reconnecting ties), Basil did three. Feasible extensions of the system to generalize over symmetries would enable him to perform five or six of the nine.

3. BACKGROUND

Historically, the automation of graphical editing tasks has progressed in two directions: interactive tools to help users with constraints and graphics-oriented programming systems. The first seeks to improve either the naturalness or the power of declaratively specified constraints, whereas the second takes a procedural approach and lets the user construct programs that express his or her intention in geometric terms. Metamouse adopts a synthesis of the two. Constraints are not declared explicitly by users but are inferred from their actions, whereas, to overcome the intractability of inference, a procedural representation is used to decompose complex global constraints into structured sequences of local ones.

3.1. The Declarative Approach

The exploitation of constraints in interactive graphics began with SKETCHPAD (Sutherland, 1963), which used numerical relaxation to resolve several types of constraints: that lines be vertical, horizontal, parallel, or perpendicular, that points lie on lines or circles; that symbols stand in vertical rows or be attached to points or lines. An interactive editing sequence typically involved new object definitions interleaved with constraint specifications. These early ideas are used in contemporary interactive drafting systems.

Two principal methods are used to facilitate high-precision interactive positioning: gravity and relaxation. *Gravity fields*, which come in various forms in graphics systems (Foley & Van Dam, 1982), are limited in expressive power and offer only a small fraction of the desired types of precision. Relaxation-based methods are exemplified by White's (1988) "human interface to least

squares," which lets users place constraints on distances and angles and then, on request through an *adjust* command, solves them using least-squares relaxation. Such systems force users to specify additional structures that are often difficult to understand and time consuming to manipulate. To combine the convenience of grids with the power of constraints, Bier and Stone's (1986) SNAP-DRAGGING technique equips users with a variety of alignment objects—such as circles of specified sizes, horizontal and vertical lines—and "snaps" points of the drawing onto them. Once used, however, constraints are discarded, and subsequent manipulations do not respect the original positioning operations.

Most constraint-satisfaction drawing aids do not allow users to define new constraints. For example, to add new kinds of constraints to the original THINGLAB (Borning, 1981), one had to write code in SMALLTALK. However, the system has since been extended to allow graphical definition of constraints (Borning, 1986). The user draws an equational network with icons that represent variables, constants, arithmetic operators, and function calls to other constraint routines. To define variables, one draws an example and labels points accordingly. Of course, the equational network requires users to have algebraic models of their problems.

Drawing is naturally procedural (van Sommers, 1984), but constraint systems are declarative. THINGLAB insists on the user specifying programs declaratively. In White's (1988) scheme, constraints are remembered, so that points, lines, and constraints can be added in any order and reapplied at any time, but it is not possible to store or manipulate a *sequence* of constraint-satisfaction problems.

3.2. The Procedural Approach

Computer drawing was originally a form of programming, with images intended for production on a plotter being expressed as FORTRAN procedures. The theoretical basis for computer graphics was provided by Descartes's conceptual breakthrough of making geometry algebraic—although the supporting technology was a long time coming. From the point of view of the user, however, Descartes's invention was a *faux pas*. A more intuitive formulation of procedural graphics was created by the ancient Greeks—indeed geometry inspired the first investigations into the very notion of a formal procedure (Preparata & Shamos, 1985). In the last two decades, constructive computer graphics have moved from an algebraic model toward purely geometric specification of graphical procedures.

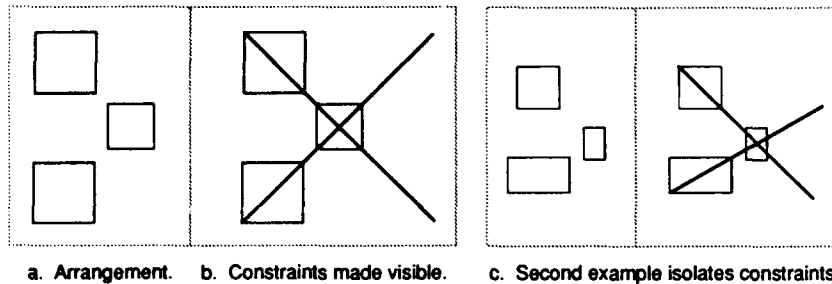
For example, LEGO specifies constraints using the traditional ruler and compass of geometric construction (Fuller & Prusinkiewicz, 1988). It provides primitives *point*, *line*, and *circle* and an operator that returns one or two points

of intersection between objects. Constructions can be automated by procedural programming. Variables are identified by naming points—in this case those returned by the intersection function. Although a graphical interface is incorporated so that users can specify procedural constructs by menu selection, users are required to identify input and output variables and control structures explicitly. Noma et al. (1988) also based a graphics language on Euclid's primitives: Users create geometric constructions by writing small programs in this language. Concrete objects are named, but abstractions (like the length of a line) are not, because the concept of a *variable* was held to be too difficult for ordinary users. Instead, the language provides a limited stacking facility to allow each primitive in the program to communicate parameters to the next.

Procedural Euclidean geometry is a viable alternative to constraint systems for specifying figures to high precision. Kin, Noma, and Kunii (1989) argued that it is superior in two respects: Constraint systems require considerable computation for large problems whereas constructive geometry is linear in the number of objects, and specifying consistent and sufficient constraints for a desired picture is a difficult task. However, the small and elegant set of Euclidean primitives, designed to provide a minimal basis for traditional "ruler-and-compass" methods of construction, does not relate well to real-life drawing—witness the fact that popular drafting programs find it expedient to offer a much richer set of pragmatically motivated objects and operations. More important, the need to deal explicitly with procedural abstractions, expressed noninteractively as text or interactively through menu selection, negates the advantages of direct-manipulation environments.

3.3. The Metamouse Approach

Instead of asking for an explicit specification of constraints or procedures, Metamouse observes the user at work and infers elementary relationships, constants, variables, loops, and branches. This is programming by example: Programs are constructed incrementally from execution traces. Some schemes that base programs on user-supplied example traces nevertheless force the user to work with programming abstractions. In TEMPO (Pence & Wakefield, 1988), users declare loops and conditional branches; SMALLSTAR (Halbert, 1984), which operates in a very general desktop domain, asks them to identify variables and their type and value range. PERIDOT (Myers, 1988) infers the range of a variable's legal values, certain spatial relations (e.g., "centered within box"), iteration over a list of objects, and the setting of active values conditional on selecting an object or mouse button. It does not infer conditional branches that affect flow of control and, hence, does not handle loops in general. Virtually no systems rely completely on automatic general-

Figure 6. Visualizing constraints.

ization; one that does, NODDY (Andreae, 1985), performs an exponentially complex induction of functions and cannot cope with errors.

Inferring a program is not easy, but inducing complex transforms from examples of input and output is completely intractable (Angluin & Smith, 1983). In effect, a demonstration decomposes the transform into a sequence of simpler ones. Drawing is inherently procedural, often systematically ordered with each step governed by very few constraints (van Sommers, 1984). Nonetheless, it is hard to induce procedures even from simple steps. Typical users do not always construct the relevant measurements and relations, but work instead by visual inspection. Their drawings may lack important construction objects. For instance, in Figure 6a, the square on the right is actually aligned with the diagonals of the squares on the left; these constraints are visualized in Figure 6b. But inferring constraints from a picture is unreliable because some relations may be incidental. In Figure 6b, the contacts between diagonal lines and the corners of the square seem to be constraints; a second example using a rectangle instead of a square (Figure 6c) demonstrates that these relations are incidental, that the constraint is between the diagonals and the rectangle's center. Curve-matching methods such as those employed in graphical search and replace (Kurlander & Bier, 1988) are successful enough to induce patterns in drawings that contain "invisible objects" or incidental relations. Moreover, examining the whole screen for implicit spatial relations would often require an infeasible number of tests and vastly expand the space of hypotheses for generalization. This is why our system restricts its attention to visible touch relations produced by explicit actions, marks touches with tacks so users can identify constraints by pointing at them, and asks users to specify complex relations by stepwise construction. To make this palatable, we adopt the coaching metaphor, which combines demonstration, observation, correction, and instruction. Our hypothesis is that, by coaching, the user will gain insights needed to present explicit demonstrations and use constructions.

Our metaphorical apprentice employs both interaction and generalization

to create a procedural model of the user's actions. It is the focus of attention of both user and system. Only local constraints involving it or an object it is grasping are examined. It incorporates an internal model of graphical constraints and asks for explanation when an action seems arbitrary (i.e., insufficiently constrained). Rules of interaction between human teachers and pupils have been formulated as "felicity conditions" (Van Lehn, 1983, p. 11), and these apply when coaching Basil too: in particular *correctness* (examples shown are assumed to be correct), *show work* (demonstrate execution rather than just input and output), *no invisible objects* (express constraints by graphical construction), and *focus activity* (eliminate extraneous actions). To help untrained teachers obey these rules, Basil builds a model of the user's actions dynamically and predicts them as early as possible during a coaching session. The metaphor encourages the teacher to demonstrate constraints and adopt an intentional stance toward the system (Dennett, 1987) rather than guess the mechanisms behind its constraint and generalization models. Whether or not it succeeds is an experimental question, but initial results are encouraging (Section 5).

3.4. Machine Learning and Generalization

To create procedures from examples of their execution, Metamouse uses some generalization techniques that have been developed in the context of machine learning. A fundamental distinction in this area is between similarity-based learning and knowledge-intensive processes such as explanation-based learning (Witten & MacDonald, 1988). Given a set of objects that represents examples and counterexamples of a concept, a similarity-based learner attempts to induce a generalized description that encompasses all the positive examples and none of the counterexamples. Typically, background knowledge is not brought to bear on the problem except insofar as it is used to delimit the space of possible descriptions that are considered (Mitchell, 1982). In contrast, explanation-based generalization methods take a single example and deduce a general rule by relating it to an existing theory (Ellman, 1989). In effect they use examples to guide the operationalization of knowledge already implicitly known, so that it can henceforth be employed more efficiently.

The Metamouse system contains elements of both types of learning. Similarity-based learning is used when forming a sequential procedural model of a sequence of actions. User-demonstrated actions are assumed to be positive examples of connections in the model, as are those actions predicted by Basil and accepted by the user. Negative examples arise from predicting actions that the user rejects. The space searched is the set of automation models consistent with the observed action sequence. A domain theory of

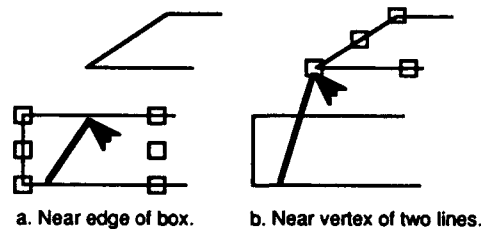
programs—which might, for example, relate programs to their effects in some formal denotational semantics—is not used to guide generalization, because of its extreme complexity.

Explanation-based learning is used in two ways. First, the user is encouraged to create an explicit explanation of his or her action sequence by employing constructive techniques to reveal hidden relationships. This differs from conventional explanation-based learning in that the user is responsible for coming up with the explanation. If he or she fails to do so, learning will not just become bogged down while the system seeks its own explanation but will fail completely. That is why we take great pains to encourage the user to demonstrate constructions explicitly. The alternative, to seek a theory that permits different constructions to be postulated and evaluated, seems too underconstrained to contemplate seriously.

The second use of explanation-based learning relates to identifying local constraints that govern actions. As explained more fully in Section 4.7, Metamouse incorporates a simple theory that distinguishes levels of significance of observed touches. The most significant are sifted out as constraints. This is a classic use of explanation-based learning to identify, via some domain theory, a subset of the currently available information that serves to identify an equivalent situation rapidly in the future. The domain theory in this case is weak in the sense that its theorems are neither universal nor rigorously derived, but encapsulate important observed tendencies in the making of drawings. If an explanation is incorrect in a given situation, the wrong constraints will be stored and the system will either be inefficient or incorrect in identifying an analogous situation in the future. If it is incorrect, the user will reject its prediction and enter the correct one, which will permit the information—but not the underlying theory—to be refined, perhaps corrected.

4. SYSTEM COMPONENTS

Basil inhabits a simple interactive graphics environment. The user teaches editing procedures by demonstration, occasionally issuing simple instructions to focus attention and correct mistaken inferences. “Teaching mode” is identical to normal editing except that the turtle icon marks the most recent mouse click’s location, and tacks mark intersecting objects. The learning module records each drawing operation at closure (a mouse click); until then, Basil waits at his last position, indicating that intermediate activity is ignored. The learning module associates objects with variables and distinguishes constraints from incidental touches. In constructing a program, it matches user actions with states and confirms a loop or a joining of branches by predicting subsequent actions. Program states are generalized actions, bound to the current situation by a constraint solver. If the constraints have no

Figure 7. Highlighting distinguished points near cursor (arrowhead).

solution (e.g., because a pool of objects for selection has been exhausted), the system forms a branch conditional on those constraints. The learning module's hypotheses (expressed in actions, icons, and menus) can be corrected through direct manipulation: by performing the desired action, clicking on a tack, or picking an alternative menu item.

4.1. Drawing Program

The drawing program A.Sq¹ resembles MacDraw (Cutter, Halpern, & Spiegel, 1987) but includes only box and line primitives. The program has three modes (each indicated by a special cursor): *create-boxes*, *create-lines*, and *edit-objects*. The user edits objects by moving iconic handles, which appear whenever the cursor approaches them, as illustrated in Figure 7. Unlike MacDraw, A.Sq provides a multilevel undo/redo.

The choice of primitives and operators has a great impact on the user's expression of constraints. A.Sq's primitive object types, auxiliary data structures, and operators are summarized in Figure 8. Points on the boundary of an object are represented in a parametric form. Any point on a line is designated by a number between 0 and 1, and on a rectangle by a number between 0 and 4. Thus, each vertex is a whole number, and each edge is a line in parametric form. Coordinates and corresponding part names are shown in Figure 9.

The basic drawing operation is to select a point on the canvas, which becomes *CurrentPoint*. According to mode, this results in selecting an object or handle, creating a graphic, or relocating a handle. The user and Basil view actions at a much higher level than A.Sq. For instance, drawing a new box involves a pair of actions for the user/Basil: in *create-boxes* mode, locate Vertex 0; then locate Vertex 2. A.Sq does the following: In *create-boxes* mode, *select-point* sets *CurrentPoint* and then activates *new-box* to allocate a rectangle having its Vertex 0 there; *new-box* temporarily sets the mode to *edit-objects*, executes *select-handle* and *translate-handle-of-object-to-point* for the handle at Vertex 2, whose

¹ Named after the protagonist of *Flatland* (Abbott, 1884).

Figure 8. Elements of the A.Sq drawing program.

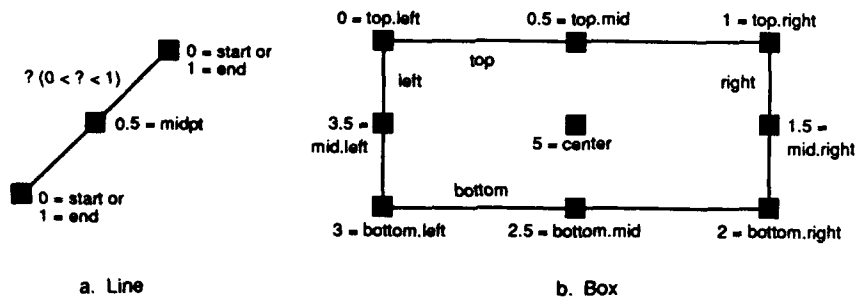
Graphic object types	Box: Specified by top.left = (x_1, y_1) , bottom.right = (x_2, y_2) ; edge coordinates $0 \leq \text{top} \leq 1 \leq \text{right} \leq 2 \leq \text{bottom} \leq 3$ $\leq \text{left} \leq 4$; handles at each vertex, midpoint of edge, and center (Coordinate 5). Line: Specified by start = (x_1, y_1) , end = (x_2, y_2) ; edge coordinates 0 . . . 1 (e.g., midpoint = 0.5); handles at start, end, and midpoint.
Auxiliary objects	Mode: {create-boxes, create-lines, edit-objects}. CurrentPoint: (x, y) location most recently selected by user. PreviousPoint: Previous value of CurrentPoint. CurrentObject: Graphic object most recently selected by user. Handle: Currently selected (activated) handle of CurrentObject. DisplayList: List of graphic objects in drawing. ActionStack: List of actions done or redone (see later). UndoneStack: List of actions undone (see later).
Drawing operators	[Note: Arguments marked « are accessed; » are set; *reference objects that are altered.] Set-mode («{create-boxes, create-lines, edit-objects},» Mode). Select-point («X, »Y, »PreviousPoint, « »CurrentPoint). Select-object («DisplayList, «CurrentPoint, »CurrentObject). Select-handle («CurrentPoint, «CurrentObject, »Handle). New-line («CurrentPoint, »*CurrentObject, »DisplayList). New-box («CurrentPoint, »*CurrentObject, »DisplayList). Translate-handle-of-object-to-point (»PreviousPoint, »CurrentPoint, »*Handle, »*CurrentObject). Delete-object (« »CurrentObject).
Action operators	Undo («ActionStack, »UndoneStack). Redo («UndoneStack, »ActionStack). Define-action («Operator, »PreviousPoint, «CurrentPoint, »CurrentObject, »*Action, »ActionStack).

interaction method in turn invokes select-point, which (because the mode is edit-objects) activates rubber banding as the user relocates Vertex 2.

At present, the drawing program is relatively simple yet rich enough to study programming-by-example issues. No conceptual difficulties are envisaged in extending the system to work with touch constraints among polygons, ellipses, and splines. We also expect to be able to accommodate new operations such as rotation, grouping, and coloring.

4.2. Basil and the User Interface

Prior to working with Basil, users skim the biosheet reproduced in Figure 10. Its concrete language and simple examples are intended to give an initial

Figure 9. Selector functions for part of an object.

conceptual model of Basil's dependence on explicit construction of spatial relations so that users might meet the "show work" and "no invisible objects" felicity conditions (see Section 3.3). Further guidance is provided by several interaction devices: the Basil and Tasks menus, the turtle icon, touch indicators, and dialogue boxes.

The *Basil menu* contains two items that toggle their contents to set the teaching mode. The command pairs are: *begin/end lesson*; and *suspend/resume watching user actions*. The latter allows the user to work out a construction or fix up the drawing without introducing irrelevant steps into the program. A third useful mode would be *begin/end block of actions to be done only by the user*; thus a program could contain arbitrary manual steps.

The *Tasks menu* contains names of procedures the user has taught Basil. It is possible to select from this while teaching so that tasks may be embedded, although the subroutine inherits no context from the current procedure.


The turtle icon has two purposes: to remind users that they are coaching and to indicate the system's focus of attention. It moves to *CurrentPoint* (figuratively, Basil's snout) after each drawing action, thus maintaining the context of relative motions.

Basil is described as near-sighted but touch-sensitive; moreover, the user needs to understand that only binary touch relations including Basil or *CurrentObject* (figuratively, the one "grasped" by the turtle) are checked. If the user intends that more remote touches play a role, he or she must move Basil to the relevant objects to check for them. To convey Basil's restricted focus of attention, the system marks touch relations it observes with tack icons, as shown in Figure 11. Black tacks mark touches that Basil considers to be important constraints; white tacks mark incidental touches. If the user disagrees, he or she may click on a tack to change it from black to white or vice versa.

Nonblocking (i.e., response optional) dialogue boxes present the system's hypothesis about Basil's path or implicit constraints. The path is shown in the prompt and in the turtle icon's heading (Figure 3g); should the user disagree,

Figure 10. Description of Metamouse given to users.

Basil



Basil is a turtle that helps you with precise, repetitive drawing. He

- remembers your actions
- predicts repeated steps
- moves objects so they touch precisely at corners, ends and centers

To teach him, choose "New task" from the "Basil" menu.
"Save task" when done.

To interrupt a lesson, choose "Take a nap".
"Wake up!" to resume.

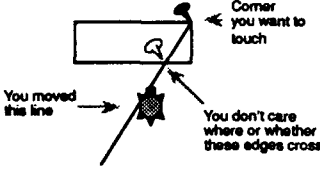
Tacks

When you select or move an object, Basil puts tacks where it touches other things, as shown here.

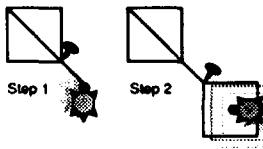
The tack's color tells whether he will make the touch happen when he does this action in future:

- black — important, make sure it happens!
- white — coincidental, don't worry about it

If you disagree with Basil's guess, click on a tack to change its color.



Tools



Basil builds pictures by plugging shapes together, like Tinker Toys.

Any shape can serve as a tool for positioning things, like the line shown here to space and align two squares.

Basil needs tools, because he learns only how things touch, not how they relate to each other at a distance.
When you make a tool, show Basil step by step how to use it.

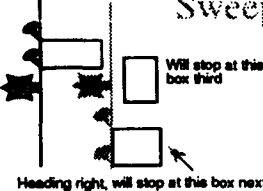
Step 1: Draw line thru first square.
Step 2: Move second square to line.
Step 3: (not shown) Delete line.

Sweep

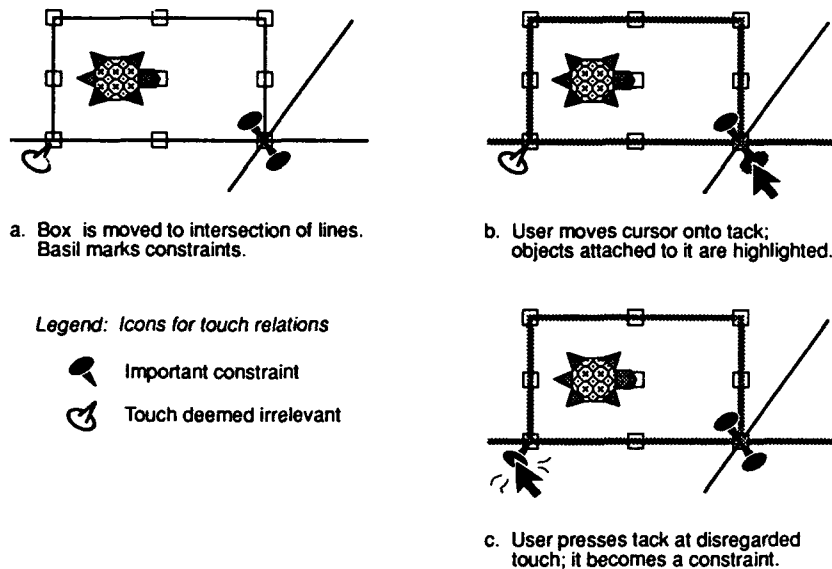
Basil can learn to scan for objects in four general directions, up, down, left, and right. His snout points where he is heading.

If you disagree, click repeatedly on his shell to rotate him.

He will search for the next object in that direction, as if sweeping with a wide broom. Draw the broom yourself (the vertical line in the figure), to be sure which one he chooses next.



he or she is prompted to turn Basil to the desired heading by tapping on him. The default implicit constraint (when no touch governs an action) is that the position is set by the user (Figure 3a). Should the user disagree, he or she may select "always here," which means constant absolute position; "this far from last point," which means constant relative position; or "relative to an object," which means that the point should have been constructed. In the latter case, Basil asks him or her to draw the construction and adjust the original object's position afterward if necessary. (The construction steps are inserted into the procedure ahead of the original action.)

Figure 11. Touch constraints are marked by tacks.

The system puts up three blocking “yes/no” dialogue boxes. When the user’s action matches some previous step, Basil requests permission to predict (Figure 3f). When performing a step, Basil displays a simplified description, stating the operator and grasped object type, and asks the coach to accept or reject it (Figures 3g–3j). A third option would be useful here: Always let the user do this step. At the end of a lesson, the user is asked whether the task should be saved: If so, he or she types in a name that will appear in the Tasks menu.

If an action is to be done by the user, Basil puts up a dialogue with the abbreviated description and response options “Done” and “No.” The appropriate object is selected or created for the user to edit. In the case of a new object, it is drawn at the same position as originally taught. When the user has finished editing it, he or she clicks “Done.”

When no prediction is performable (as in Figure 3k), Basil displays an advisory message asking the user to demonstrate the default action.

The system has several display options, intended mainly for use in our research. The turtle icon may be shown at all times during a lesson, only when Basil is predicting an action, or not at all. Tacks of either color may be shown or hidden.

4.3. Overview of Learning Module

Figure 12 depicts the learning system. During a demonstration cycle, in the top half of the diagram, the user performs actions the system analyzes and appends to the program. When a user action matches some existing program

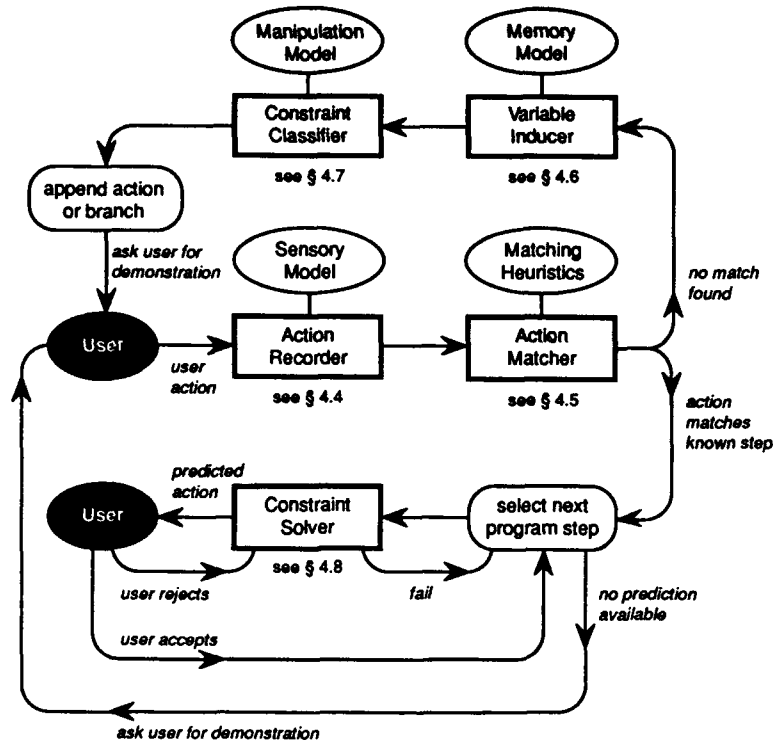
step, the system enters a prediction/performance cycle, shown in the diagram's lower half. Basil executes program actions until the user objects or no step is performable. When this happens, Basil asks the user to take over and the system returns to the demonstration cycle. The learning algorithm is summarized in the following two paragraphs.

Demonstration Cycle. The Action Recorder observes each edit performed by the user, noting its operator (e.g., *drag-handle*) and its observable postconditions, which are touch relations involving CurrentObject. The Action Matcher searches the program for a step whose goal is met by the new action's postconditions. If such a step is found, the system enters a prediction/performance cycle (see next paragraph). Otherwise, the new action is analyzed as follows. First, the Variable Inducer finds or allocates variables for objects in touch relations. The action is then passed to the Constraint Classifier, an explanation-based generalization module that uses domain knowledge about the current operator in order to isolate those touches that would constrain its parameter values. Thus, the action's goal is identified as a subset of its observed postconditions. Finally, the action is passed to the Program Manager (not shown in Figure 12), which appends it to that branch of the program currently being taught. Should the user's action immediately follow a rejected or failed prediction (see next paragraph), the step is added as a new branch.

Prediction/Performance Cycle. When the Action Matcher finds a program step whose goal is met by the user's action, the Program Manager does two things. First, it makes a link from the previous user action (the end of the new branch) to the matched step; this link is subject to confirmation (to be discussed later). Second, it updates the program state, marking the matched step as the one most recently performed and rebinding variables to objects in the user's action. The system then selects the next program step for execution, from among a preference-ordered list of alternatives. It predicts alternatives until both the constraint solver and the user accept one, or until none remains. In the former case, the program state advances, variables are updated, and execution continues from the accepted step's successors. In the latter, the system returns to the demonstration cycle, with new actions to be appended after the last accepted step. If at least one prediction was accepted during this cycle, the new link is confirmed; otherwise it is deleted, the match is canceled, and new actions will be appended to the branch from which the link was tried.

Note that an action is performed for the user only if the constraint solver can instantiate its goal. Thus, failure is an implicit branching condition, which is used to advantage in terminating loops. Our current implementation does not infer explicit conditional tests on some subset of postconditions other than the goal. Therefore, Metamouse cannot learn to choose an action based on Basil's current sensory information. Nevertheless, it can learn "if-then"

Figure 12. Components of learning system.



constructs, provided the user teaches the conditional test as (the goal of) an explicit action.

When the user rejects an action, it is moved to the end of the list of current alternatives so that it will be predicted again only if all others fail to pass the constraint solver. This tactic finesses the problems of debugging. It avoids asking the user whether an action is erroneous in all situations, has an overly general goal, depends on a conditional test that Basil cannot learn, or is merely inappropriate in the current usage of this procedure. It also skirts the problem of determining the extent of a bug, that is, what substructure of the procedure ought to be deleted. A simple extension would make action rejection a more reliable debugging method. There should be available a third response to predictions: "Don't predict this alternative again."

4.4. Action Recorder

Each of the user's editing actions is recorded, together with its context—a part of A.Sq's state isolated by Basil's focus of attention. (Recording only part

of the state constitutes implicit generalization.) All actions of the current lesson are remembered in sequence in an *ActionTrace*, from which the matcher builds a program (Section 4.5). Each program step references the actions from which it was generalized.

An *ActionRecord* has three components: *Operator*, *Parameter*, and *Results*. For instance, when the user drags the sweepline upward to a box in Figure 3d, the following is recorded:

```
Operator:  drag-handle
Parameter: Handle = Line041.midpt
Results:  CurrentPoint = (240,130)
          grasp (Line041.[0.52])
          touch (Line041.[0.07] : Box057.[3.00])
          touch (Line041.[0.35] : Box057.[2.00])
          touch (Line041.[0.38] : Line040.[0.22])
```

The *Operator* is a composite of those defined by the drawing program, so that it corresponds with the granularity of actions as seen by the user—one action per mouse click. Thus, the five basic operators are: *locate*, which places a new graphic's first point; *select*, which picks a new *CurrentObject* or active *Handle*; *draw-line* and *draw-box*, which sweep out new objects; and *drag-handle*, which translates the active *Handle* to a new *CurrentPoint*.

Parameter identifies *CurrentObject* and the currently active *Handle* (if Basil is dragging or drawing something).

The *Results* are the action's observed postconditions, comprising the new mouse location (*CurrentPoint*) and a list of *TouchRelations* occurring in Basil's immediate vicinity. A subset of these, chosen by the constraint classifier (Section 4.7), is assumed to be the action's goal in the sense of Fikes and Nilsson (1971)—a conjunction of results that must hold in every instance. Restricting the sensory focus of attention reduces the time spent checking for touches and simplifies both the inference and run-time evaluation of the goal.

To assemble *TouchRelations*, the recorder scans A.Sq's *DisplayList*, selecting graphics that touch either Basil or *CurrentObject*. A *TouchRelation* is defined as:

$$\text{touch} (\text{Object}_1.\text{Part}_1 : \text{Object}_2.\text{Part}_2),$$

where, for $i = 1$ or 2 , Part_i is an edge coordinate in Object_i (see Figure 8) and Object_i is the graphic's address in *DisplayList*. Object_1 is either Basil or *CurrentObject*. If Object_1 is Basil, then Part_1 is 0 (his snout). If Object_2 is *CurrentObject*, the touch relation is distinguished as *grasp*($\text{Object}_2.\text{Part}_2$)—The difference between touch and grasp proves important when inducing constraints.

If the user undoes an action, it is removed from the trace and reference to

it is removed from the corresponding program step. If the step represents no other actions in the trace, it is replaced with a dummy node, which in effect links all of its predecessors to its successors.

4.5. Action Matcher

The action matcher searches the program for a step that is equivalent to the one just performed by the user. A step is equivalent to an action if it can produce the same effects in the same situation: in other words, if the program could have predicted the user's action had it been told to do so.

A program learned by Basil is a directed graph of *ProgramSteps* with no restrictions on connectivity: It may contain arbitrary multiway branches and loops with jumps into or out of their bodies. An example is shown in Figure 5.

Each *ProgramStep* has three components: *Predecessors*, *ActionGenerator*, and *Successors*. The first and third are lists of *ProgramSteps* that precede or follow the given *ProgramStep*. Successors are ordered according to the priority at which they may be predicted. *ActionGenerator*, a generalized *ActionRecord* (Section 4.4), has three parts: *Operator*, *Parameter*, and *Constraints*. For example, Step 5 of the align boxes task, in which the user grasps the sweepline (see Figures 5 and 3c) is:

```

Predecessors:    {Step4, Step8}
Successors:      {Step6, Step9} Note: Do Step9 if Step6 fails
ActionGenerator: Operator = select
                  Parameter = nil
                  Constraints = grasp ([L2 = Line041].[P9 = 0.5])

```

Operator is one of the five actions listed in Section 4.4, and *Parameter* specifies the object and handle in Basil's grasp prior to the action. *Constraints* make up a set of touch relations or position specifiers that must hold after executing *Operator* with the given *Parameter*. Constraints are generalized touch relations, where variables (e.g., L2, P9) stand for object and part identifiers. They are instantiated and checked by a constraint solver (Section 4.8).

A *ProgramStep* matches an *ActionRecord* if the following conditions hold (they are checked in order for quick rejection of mismatches). First, *Operators* must be the same. Second, the *ActionRecord* must contain at least as many *TouchRelations* as the *ProgramStep* has *Constraints*. Third, *Parameters* must agree: That is, the *ProgramStep*'s object and part selectors (see Section 4.6) must generate the object address and part coordinates specified in the *ActionRecord* (for the example just shown, they are nil). Fourth, each *Constraint* must have a corresponding *TouchRelation* such that its selector

functions evaluate to the latter's object address and part coordinate. (Part coordinates match if they lie within a defined tolerance.)

Some Constraint selector functions that search for objects must be evaluated with respect to the A.Sq environment as it was just before the user's action. This is accomplished by temporarily restoring Basil and CurrentObject to their previous coordinates, without updating the display screen.

In effect, action matching is solving for constraints where only one potential solution, the demonstrated action, can be checked. For instance, Figure 3f matches Step5 of Figure 5: Both Operators are select; both Parameters are nil; and Figure 3f's Results include a TouchRelation corresponding to the only Step5 Constraint, grasp ([L2 = Line041].[P9 = 0.5]).

A user action matches a *ProgramStep* with a constant position constraint if its resultant CurrentPoint lies within several pixels of the constant. A step whose constraint is an input position will match an action with no touch relations, regardless of where its CurrentPoint lies.

The matcher can be parameterized to search forward or backward, breadthfirst or depthfirst, starting from the graph's entry point or from the last accepted step, and to stop at the first match or to find all matches. For the evaluation study (Section 5.1), it was configured to search backward depthfirst from the last accepted step until the first match was found.

4.6. Variable Inducer

For Metamouse to apply the same task to different objects, as when iterating over a set, it must use variables in constraint expressions. Variables may be thought of as representing roles (Rich & Waters, 1988) such as "X: the object in Basil's grasp three steps before this one," or "Y: a box lying above Basil's current location and not previously used in this operation." Some aspects of a role are implicit in a variable's context (the action and touch relations in which it was defined), but other criteria, such as whether the object has been used before, are expressed by *Selector* functions. The variable inducer creates placeholders for objects and parts in touch relations. These are used by the constraint solver to instantiate a program step.

A *Variable* is local to a ProgramStep and often appears in several *Constraint* records. Each has four components: *Name*, *Type*, *Selector*, and *Bindings*. For instance, here is the variable for the box found by the sweepline in Figure 3g (the second iteration of align boxes):

```
Name:      B1
Type:      box
Selector:   find-novel-object ([Box057], box, Upward)
Bindings:   [Box061, Box057]
```

Figure 13. Selector functions.

<i>create</i> (<i>ObjectType</i>)	This step creates (draws) the object.
<i>use-value-of</i> (<i>Var</i>)	The object appears as the binding of some variable <i>Var</i> .
<i>find-named-part</i> (<i>PartName</i>)	Returns the edge coordinates of the named part.
<i>find-novel-object</i> (<i>PreviousBindings</i> , <i>ObjectType</i> , <i>Path</i>)	There is no other reference to the object in the current binding environment—It is encountered for the first time.

The *Name*, B1, is a globally scoped symbol. *Type* is one of {*box*, *line*, *handle*, *edge*}. *Selector* is the function to be called by the constraint solver (Section 4.8) when a new value is required. Figure 13 lists the four functions currently in use: The first two are common to both objects and parts; the third, *find-named-part*, is a table-lookup from part names to edge coordinates (see Figure 9); the fourth, *find-novel-object*, chooses a graphic of this variable's type that was not bound to it before (this prevents reediting an object that may have been moved into the range of search). It has an optional argument, the direction of search, in case the constraint classifier decides *Path* is relevant.

Bindings is a stack of the variable's values—object addresses or part coordinates. Previous bindings are remembered just in case *Selector* requires them, as in the *find-novel-object* example just shown.

Algorithm. Given an *ActionRecord*, the variable inducer assigns each object and part value in each *TouchRelation* to some local *Variable*. First, ensure a 1:1 mapping of objects and *Variables*. If an object address is already assigned to some *Variable*, use that *Variable* again. If a part coordinate and its containing object are both already assigned, use the existing part *Variable*. Otherwise create a new *Variable*; initialize its *Name*, *Type*, and *Bindings*; and then find an appropriate *Selector* according to the rules given later.

If an object was drawn by the current action, its *Selector* is, *create* (*ObjectType*). Otherwise, scan backward through the action trace. If the value is the current binding of some variable *X* in a previous action, then the *Selector* is, *use-value-of* (*X*). The default part *Selector* is *find-named-part*, where the name is that corresponding to the part coordinate in the *TouchRelation* (Figure 9). In the present implementation, it is assumed that a line is directed, so the selector for an endpoint indicates whether it is the start or end of the line. The default object selector is *find-novel-object*, which means that the constraint solver will scan along a specified direction (*path*) for an object not used previously as a binding for this variable.

4.7. Constraint Classifier

The key to generalizing an action is to distinguish those touch relations that are intended from those that are incidental. We call the former *constraints*.

Isolating constraints is, of necessity, a heuristic procedure. One approach is to gather multiple examples and choose those touches that occur in every one—This is similarity-based generalization. To minimize the number of examples the user gives, we took another approach—explanation-based generalization. The constraint classifier examines each *TouchRelation* of the current *ActionRecord* and assigns it to one of eight levels of significance. The most important touches are selected as constraints, and the rest are deemed incidental. If insufficient constraint is found, the module may mark direction of movement as an additional criterion or initiate a dialogue with the user regarding implicit constraint (see Section 4.2 and Figures 3a–3b).

A *ConstraintRecord* has three parts: *TouchRelation*, *Level*, and *Classification*. For instance, when the sweepline meets the first box (Figure 3d), the following *ConstraintRecords* are added to the action (note that variables have been inserted already):

<i>grasp (L2.midpt)</i>	<i>Level: Trivial</i>	<i>Class: Incidental</i>
<i>touch (L2.? : L1.?)</i>	<i>Level: Sustained</i>	<i>Class: Incidental</i>
<i>touch (L2.? : B1.bottom.left)</i>	<i>Level: Weak2</i>	<i>Class: Constraint</i>
<i>touch (L2.? : B1.bottom.right)</i>	<i>Level: Weak2</i>	<i>Class: Constraint</i>
<i>Path (Upward) is a Constraint</i>		

Because the grasp cannot be changed by drag-handle, it is not significant. The fact that sweeplines and guidelines cross, just expressed as touch (L2.? : L1.?), is sustained throughout the action is, hence, judged less significant than contacts between sweepline and box, which result from the action. Only the most significant touches are chosen as constraints. In addition, the upward path is taken to be a constraint on the choice of B1.

Method. The selection of constraints from observed touch relations is a three-stage process. First, each touch is classified according to whether the action caused it, altered it, or had no effect on it. Second, the touch is assigned a level of significance according to the type of effect and the number of variables that take their value from a set of multiple options. Third, all touch relations are ranked by significance, and those at the highest level are selected as constraints.

1. Type of Effect. To determine how the action affected a given touch relation, the classifier consults the decision table shown in Figure 14. A *Sustained* touch holds true throughout the action: That is, object and part identifiers remain the same, although part coordinates may change (as when the sweepline slides along the guideline). An *Effected* touch occurs as a result of the action (e.g., the touches between sweepline and box). A *Trivial* touch is “Sustained by definition,” that is, under no circumstances could it cease to hold as a result of the action (e.g., grasping a handle as it is dragged). An *Unaffected* touch must have held prior to this action, even though not sensed

Figure 14. Decision table classifying ways a touch relation results from an action.

Operator	Locality of touch	Both parts stationary	Relation is changed	Type of effect
locate or select	grasp or	—	no	Sustained
	direct		yes	Effected
	indirect	—	—	Unaffected
create-line or create-box or drag-handle	grasp	—	—	Trivial
	direct	—	no	Sustained
			yes	Effected
	indirect	yes	—	Trivial
			no	Sustained
		no	yes	Effected

beforehand (e.g., when Basil moves to grasp an object, any touch relations it already has with others are Unaffected).

The decision rules check the Operator, locality of touch, whether both parts remain stationary, and whether the touch relation persists. Locality of touch distinguishes grasp (i.e., between Basil and CurrentObject), direct touch (between Basil and some other object), and indirect touch (between CurrentObject and some other). Direct touches occur when Basil locates the start of a line or box at some point on another object, or moves to grasp at a point where several objects intersect.

2. Level of Significance. To decide a touch's level of significance, the classifier consults the rules given in Figure 15. The only interesting rules concern Effected touches, where the level of significance depends on the number of free variables. A variable is free if its value is chosen from a set of alternatives. A TouchRelation has at most three such variables: Object₂, Part₁, and Part₂. An object variable is free if its Selector scans the DisplayList, as does find-novel-object for B1 in the previous examples. Given the way TouchRelations are defined, this may be true of Object₂ but not of Object₁. A part variable is free if its Selector returns a range of parameter values, as in find-named-part(?) for L1, which returns the edge 0 . . . 1. A

Figure 15. Decision table for the level of touch constraint.

Type of effect	Free variables	Level
Effected	$n = 0$	Determining
	$n > 0$	Weak n
Unaffected	—	Unaffected
Sustained	—	Sustained
Trivial	$n = 0$	Trivial
	$n > 0$	ERROR

contact between a vertex and an edge has one such degree of freedom; contact between two edges has two.

In decreasing order, the levels of significance are: *Determining*; *Weak 1,2,3*; *Unaffected*; *Sustained*; and *Trivial*. This ordering reflects the ability of a touch to limit a set of positions derived by the constraint solver (Section 4.8).

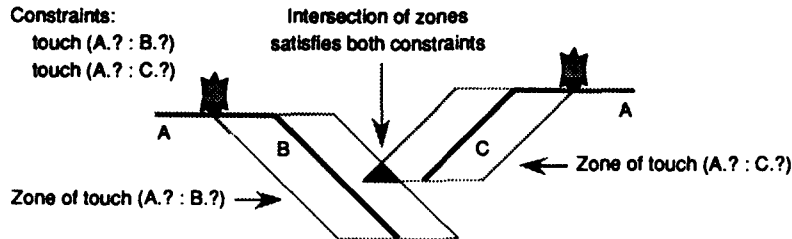
A *Determining* touch is Effected by the action and involves no options—It chooses a specific object, part, and point of contact for each item. Indeed, it specifies exactly the position Basil must occupy after the action. In Figure 3e, grasp (B1.center) involves a predetermined object and a single point of contact, hence it is Determining.

Weak touches are Effected by operators other than select and have one, two, or three options. In the previous example, touch (L2.? : B1.bottom.left) results from drag-handle applied to L2, with options in the choice of box for B1 and its point of contact along L2.

Unaffected and *Sustained* touches are assigned to levels of the same name regardless of options. They are considered of low significance because typically they do not limit constraint solutions as much as the higher levels. *Trivial* touches can have no effect on constraint solutions.

3. Selection. Having assigned each touch to a level, the classifier then selects all those at the highest level present as Constraints and marks the rest *Incidental*. If there is no Determining constraint, the Path is made a Constraint with the caveat that the solver may have to relax it. Should there be no touches at a level higher than Unaffected, the classifier signals inadequate constraint. In effect, the solver requires user intervention to produce a specific result, so a default Constraint, “ask the user,” is adopted.

An Incidental touch that instantiates an object variable (i.e., a relation containing a variable whose selector is find-novel-object) will be promoted to

Figure 16. Intersection of solution zones for competing constraints.

a Constraint if some future action refers to that object. The classifier signals the interface to display appropriate interaction devices for Constraints, Incidental touches, and Paths, as described in Section 4.2.

Obviously, the classifier's judgments cannot be guaranteed correct. For instance, in the alignment task variant (Section 2.4), the grasping of a box is governed by a Determining constraint, yet the Unaffected contacts between a box and its tie-lines should be used to restrict the choice of box to one having a tie-line at the left edge. This points to the need for similarity-based generalization (which is not used) and user intervention (which is supported).

4.8. Constraint Solver

Solving constraints is the process by which a predicted ProgramStep is realized as an action with specific values for object and part variables and for CurrentPoint (i.e., Basil's new location). Figure 16 shows what happens when a Line A is to be moved so that any part of it touches both Lines B and C. The solution zone is a convex two-dimensional area within which CurrentPoint may lie, such that the stated touch relations hold. A set of constraints is solved by intersecting their zones: This involves trying different combinations of permitted values for variables (consistent across constraints) and intersecting each region with the next until the result is empty (failure) or no constraints remain (success). If no combination of variable values yields a nonempty solution, the action is not performable (see Section 4.3). Otherwise the solver chooses a point within the solution using additional criteria (see the following).

Algorithm. The solver recursively processes a list of Constraints. The initial zone is the entire drawing. For each Constraint C, it tries alternative values of variables "owned" by C until it finds a combination for which C's zone overlaps both the area already computed and the solution to the remaining Constraints given the current variable bindings. If no result is nonempty, the solver returns to the previous Constraint and tries alternative

bindings there. It reports failure if none remains for variables of the first Constraint; otherwise it returns a nonempty solution zone.

A Constraint owns variables that occur in no earlier member of the list. Only these may be re-bound; otherwise the previous zone would be invalidated. Combinations are generated, one at a time, by rebinding one variable and reinitializing those for which no alternative values remain. Because the first object in a touch relation is either Basil or CurrentObject, a Constraint has at most two variables (object and part) to rebind. A variable is bound by its Selector function, which chooses a value from the DisplayList (objects) or object description (parts). As noted in Section 4.6, Selectors impose their own constraints on variables; for instance, use-value-of (Var) permits only one value. Find-novel-object uses a Path criterion to select only DisplayList items whose location is in a certain half-plane relative to Basil.

For instance, in Figure 3g, Basil must solve the following pair of constraints with the given Path criterion:

Path (Upward)

C1: touch (L2.? : B1.bottom.left)

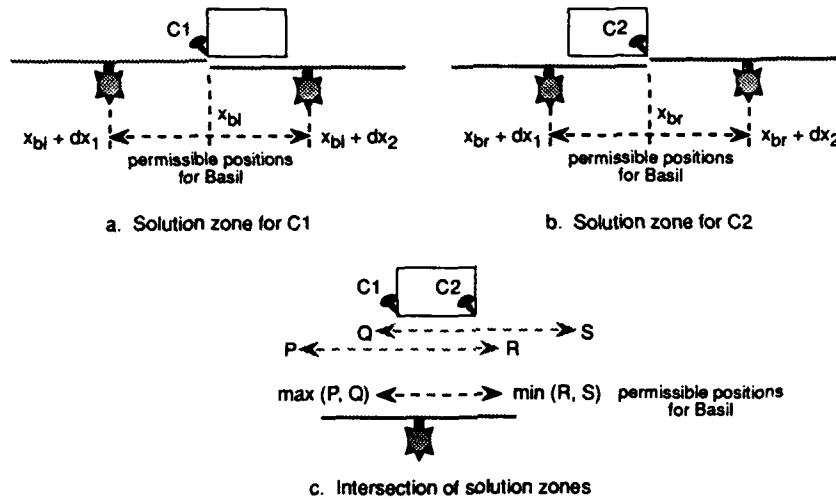
C2: touch (L2.? : B1.bottom.right)

When processing C1, the solver binds the variables it owns: L2, L2.?, B1, and B1.bottom.left. Because L2's Selector is use-value-of (L2), its value is not changed. B1's Selector is find-novel-object, so it is assigned the nearest box along the vertical dimension (given by Path). The part variables are given parameter ranges (0..1 and 3, respectively) according to definitions in Figure 8. When processing C2, the solver can bind only one variable, B1.bottom.right.

A Constraint's zone is a polygon whose vertices are extreme positions that Basil might occupy (e.g., see Figure 16). Consider the touch relation touch ($o_1.p_1 : o_2.p_2$), where o_i is Basil or CurrentObject. Each part p_n can be thought of as having one or two vertices (it is a handle or an edge). Basil is at offset dx_v, dy_v from each vertex v of p_1 . For each vertex w of p_2 , the solution zone has one or two vertices at $(x_w + dx_v, y_w + dy_v)$.

The solution zones for the example from Figure 3g are detailed in Figure 17. Basil is at L2.midpt; hence, if $L = \text{length}(L2)$, then $(dx_1, dy_1) = (-L/2, 0)$ and $(dx_2, dy_2) = (L/2, 0)$. For C1, p_2 has one vertex B1.bottom.left whose coordinates are (x_{bl}, y_{bl}) . Thus, C1's zone is a line extending from $(x_{bl} + dx_1, y_{bl})$ to $(x_{bl} + dx_2, y_{bl})$. Similarly, C2's zone is a line from $(x_{br} + dx_1, y_{br})$ to $(x_{br} + dx_2, y_{br})$.

Because solution zones are (necessarily) convex polygons, a fast clipping algorithm developed by Sutherland and Hodgman (1974) is used to intersect them. Zones that are single points, line segments, or vertical or horizontal lines are treated as special cases to further speed intersection calculations. In Figure 17, where the zones of C1 and C2 are horizontal lines, their

Figure 17. Calculating solutions for constraints between a swepline and a box.

intersection is a line from $(\max(x_{bl} + dx_1, x_{br} + dx_1), y_{bl})$ to $(\min(x_{bl} + dx_2, x_{br} + dx_2), y_{bl})$.

Having processed all constraints, the solver chooses a single point within their intersection as the final solution. If *Path* is a constraint, it chooses the point nearest to but within the half-plane ahead of Basil's present location. Otherwise, it chooses the zone's centroid. In Figure 17, *Path* is relevant and the nearest point is directly above Basil, $(\text{CurrentPoint}.x, y_{bl})$.

The constraint solver always terminates. The combinatorial component's complexity is proportional to the product of the number of variables and the number of feasible values. The numerical component, invoked for each iteration of the combinatorial one, is linear in the number of constraints. Selector functions are linear in the number of possible values but in the worst case could be invoked on each iteration of the combinatorial solver. This suggests that potential values should be ordered (e.g., *DisplayList* objects could be sorted along the search path) and that feasible values should be memorized.

5. EVALUATION

We now establish that *Metamouse* actually learns procedures from example execution traces, and we summarize the results of a study of the extent to which inexperienced teachers understand its behavior. More detailed usability studies will be carried out when our new implementation has been thoroughly debugged.

Figure 18. Performance of the learning system on three tasks.

Task	Actions Performed					Size of Program
	Trace	Total	Predict:Inputs	%Accepted	Rejected	
Stove	1	10	0	0	0	11
	2	10	10:1	90	0	11
Sorting	1	32	18	56	1	21
	2	38	38:2	100	0	22
Align boxes	1	20	8	40	0	13
	2	24	24:2	100	0	13
. . . variant	3	32	18:2	56	3	28
	4	32	32:2	100	0	28
. . . original	5	24	24:2	100	0	28

5.1. Performance of Tasks

Concepts learned by Metamouse cannot be judged correct or incorrect because users demonstrate only enough to solve the problem at hand. However, their coverage, robustness, and complexity may be examined during coaching sessions. *Coverage* is measured as the ratio of actions correctly predicted to the total number performed by both user and apprentice. The *rate of learning* is measured as the increase in this ratio from one trace to the next, or between iterations of a loop. *Robustness* is measured as the ratio of incorrect predictions (i.e., ones rejected by the user) to total predictions. *Complexity* is related to the number of edges (i.e., transitions between actions) in the program graph.

Another important performance measure is actual running speed—specifically, delays introduced by matching the user's latest action and by solving for constraints when predicting. Although we have not done detailed timings, we find that, for relatively small tasks like those described here, the system (running on a Sun SPARCStation) responds in real time.

To establish that Metamouse can infer constraints and procedures from graphical constructions, it was taught the tasks described in Section 2 using the same procedures. This study does not purport to show that typical users of Metamouse would produce demonstrations with similar constructions—although we believe this to be the case.

Each task was demonstrated once and then invoked several times on different data; the demonstrations were free of incorrect or extraneous action. It was found that the system quickly achieved competence and constructed simple models. Figure 18 summarizes performance data. It compares the total number of actions in each trace with the number correctly predicted by Basil,

also shown as a percentage of the total. The count of predictions includes the number of user inputs, noted beside it. The number of predictions the user rejected is also shown. The size of the program graphs is given as the number of edges.

Stove. The stove editing task is a simple sequence of actions. No predictions can be made during the first trace, as it contains no repeated actions. All actions in the second trace are predicted correctly. One input from the user to move the stove is required.

Sorting. For the first trace of the sorting task, four boxes were used; for the second, five boxes were used. By inducing loops, the system was able to perform 56% of the actions in this task the first time the user performed it. Automation increased to 100% on the second trace, with two inputs to edit the spacer.

Different construction tactics for sorting and spacing can be used: Their effectiveness depends on exploiting Basil's model of constraints (of which the user is well informed) and remaining within the system's inferencing limits (of which the user is not). "Violations" of the latter condition can have unpredictable results. For instance, suppose the swepline is eliminated when sorting. This makes no real difference because Basil orders the find-novel-object selections by distance along the axis of Path. It would help the user understand Basil better, however, if a swepline were generated automatically when using this selector.

A more serious misunderstanding occurs when the first box is placed to the left of the spacing tool, with the remainder to the right. Basil predicts that the second box should go to the left like the first. When the user rejects this, a branch is formed that gives priority to putting the currently selected box at the right. On the next invocation, Basil puts the first box at the right: When the user rejects this, Basil tries an alternative prediction, which is accepted, causing the priorities to be reversed again. Thus, on every invocation, Basil handles the first two boxes incorrectly (unless the user accepts Basil's putting the first box at the right!). This problem arises from using a fixed number of action matches (viz., one) to induce a loop. We have created a new learning algorithm that is capable of extending the match context post hoc so that such loops can be split; this will be incorporated into the next version of Metamouse.

Aligning Boxes. Five traces of the alignment task were produced. The first involved three boxes as in Section 2.3; the second was run on four boxes; the third and fourth introduced and repeated the variant described in Section 2.4; and the final trace was a repetition of the second. Basil was able to learn the variant and yet retain the ability to do the original task. In the first trace,

the system predicted the second and third iterations of the loop, or 40% of the work. In the second trace, it processed all four boxes, with two inputs for the guideline's endpoints. The third trace adapted the alignment procedure to the task illustrated in Figure 4; 56% of its actions were predicted, and steps demonstrated by the user introduced 15 new state transitions (see Figure 5). During training, Basil made three faulty predictions. On the first iteration, Basil went to the sweepline after editing the box; the user rejected this and edited the tie-line. On the second iteration, when the sweepline touched two boxes, Basil picked the one on the left, but the user rejected this and chose the one on the right. After the final iteration, Basil predicted the end of the task, but the user edited the vertical tie-lines.

Because new actions are given priority over old ones, Basil was able to repeat the variant in Trace four without error. On the other hand, because actions are predicted only if their constraints can be solved, Basil was able to repeat the original task in Trace five without making irrelevant predictions concerning nonexistent tie-lines.

5.2. Evaluating Interaction

A critical aspect of a learning system is that the teacher must understand its behavior (MacDonald & Witten, 1987). The suitability of the Basil metaphor is measured as the ease with which teachers learn to predict what it will do. This has been studied in two questionnaire-based experiments (reported in more detail by Maulsby, James, & Witten, 1989). The first, a pilot study without controls, was intended to establish the viability of a questionnaire. The second introduced controls on the amount of prior knowledge subjects were given regarding the metaphor and also measured correlations with previous computing experience. The results of both experiments show that even without live interaction Basil's behavior is largely self-explanatory or easily rationalized. They do not, however, directly address the issue of creating procedures by coaching Basil.

Pilot Study

In the pilot study, subjects were given a brief description of Basil (an earlier version of Figure 10) and then asked to work through a self-study guide. Typical questions depict a situation and ask the subject to predict Basil's response. Correct answers were provided after each page of questions to simulate system feedback. A sample page is shown in Figure 19: The subject is asked to state the discriminations he or she believes Basil would make between touch relations. The experiment was run with eight volunteer subjects who worked at their own pace.

If the metaphor were difficult to understand, one would expect numerous

Figure 19. Sample page from questionnaire, with correct answers.

Basil pays attention to certain kinds of sensory feedback in order to distinguish one situation from another. For each pair of frames below, indicate whether or not Basil would distinguish the two situations.

		<input type="checkbox"/> Same <input checked="" type="checkbox"/> Different
		<input checked="" type="checkbox"/> Same <input type="checkbox"/> Different
		<input checked="" type="checkbox"/> Same <input type="checkbox"/> Different
		<input type="checkbox"/> Same <input checked="" type="checkbox"/> Different
		<input checked="" type="checkbox"/> Same <input type="checkbox"/> Different

errors in early questions, with at best a slow improvement. If completely obvious, one would expect near-perfect performance from the beginning with no degradation. The results show excellent performance initially, with occasional mistakes and difficult spots after which near-perfect performance is restored. It was concluded that the “superficial” aspects of the metaphor—namely, the rules that distinguish parts of objects and types of direct touch—are easily understood, whereas deeper aspects—the rules that govern action matching and prediction—are less obvious but learnable.

Controlled Study

A follow-up study—with students of architecture and industrial design (giving 16 responses) and of first-year computer science (giving 20 responses) as subjects—investigated two hypotheses: (a) that the amount of explanation of Basil's behavior given prior to examples of it does not significantly affect its predictability and (b) that prior experience with computer systems does not significantly affect comprehension of the metaphor. The first hypothesis was not disproven, indicating that the metaphor communicates essential aspects of the system's operation intuitively. The second was contradicted, but it was found that the most useful types of experience were of graphical interfaces and drawing programs, as opposed to computer programming and spreadsheets.

Several controls were introduced. First, the introductory material was varied. One version of the questionnaire contained the full description of Basil (Figure 10). A less informative version came with a two-page worked example of Basil learning a simple task. A minimal version provided a meagre one-paragraph explanation of terms used in the questionnaire. This variation had no significant impact on subjects' overall scores or on scores for the first page of the questionnaire. Second, the order of questions was varied to simulate interaction rather than guided study; this was found to have no significant effect on performance. Third, some subjects were given no answer key (i.e., no feedback); this control group was eliminated due to lack of response.

6. FUTURE WORK

A project that combines machine learning, constraint solving, and graphical interaction affords many avenues for further research. Some of these relate to improvements in and evaluations of Metamouse, others to the wider problems of programming by demonstration. We consider the following projects most important:

- Perform usability studies on Metamouse to determine whether novice users can program tasks by demonstration using graphical constructions.
- Perform ergonomic studies, measuring improvements in task execution time achieved through programming by demonstration.
- Develop a richer set of object selector functions and provide an interface similar to that for constraints (marked by tacks), so that the user can see and alter Basil's hypotheses.

- Implement the formation of conditional branches based on the preconditions of actions.
- Augment the system with similarity-based learning of constraints and selector functions to reduce spurious branching and increase predictions.
- Extend A.Sq to include circles, polygons, and rotation; this will necessitate changes to the constraint solver because solution zones will no longer be restricted to convex polygons.
- Introduce orientation-dependent naming of object parts (e.g., leftmost end of line); use similarity-based learning and allow direct user access to choose the appropriate selector.
- Induce certain common spatial relations (e.g., alignment along a major axis) so that construction is not always required.
- If Metamouse infers a spatial relation or an ordering of selections along a path, express it by generating a tool (like a sweep line) so that users will learn from Basil how to make appropriate constructions.
- Investigate the use of a different modality (e.g., voice) for the dialogue with Basil; this would enhance the separation in the user's mind between the application and the apprentice.
- Develop a cleaner and more elegant theory of constraints in a drawing world, without sacrificing predictive power.
- Provide a clearer separation between the programming-by-example method and the application domain, and test the method's viability in other domains.
- Develop a more layered approach to implementation that reduces its complexity and general unwieldiness.

We are beginning to address several of these issues within the framework of an "instructible system"—one that combines inference from examples with direct instructions from the user.

7. CONCLUSIONS

The nature of Metamouse raises several important questions. The system is designed to build a predictive model of human performance by conjecturing intentions behind isolated actions. It is illuminating to consider what kinds of procedures Metamouse can and cannot learn. In a trivial sense, the system is "Turing complete": One can teach it to emulate any finite-state

automation, including the control for a Turing machine, and give it access to a graphical memory of linear structure and arbitrary size. The issue then becomes one of teachability rather than learnability: what users find natural to teach rather than what can in principle be taught.

Some tasks cannot be represented without creating unnatural objects to support the computation—A good example is counted loops, where an external graphical counter can in principle be constructed but is tedious to create, update, and test. Other tasks may present teachers with an unreasonable mental burden, for example, reference to higher order indirect touches and demonstrating many different cases by iterating over each one in turn. Further problems arise from the difficulty of structuring programs and the fact that subprocedures are not supported. Perhaps it is reasonable to assume an upper limit on the complexity of any program that it is worth teaching a system that does not provide an externalized, written record.

Metamouse constructs nondeterministic procedures that use constraints and, as a last resort, recency to disambiguate alternative branches at run-time. This has striking benefits when debugging, extending, and reusing procedures. It also finesses the problem of prematurely formed loops, formed when a sequence includes matching subsequences that are long enough to satisfy loop confirmation.

The preliminary human factors experiments on Metamouse's usability may lead to modifications of the user interface. In fact, subjects of the pilot experiment had difficulty predicting Basil's sensory discriminations and classification of constraints, and this led to a more telling graphical representation for Basil's sensory feedback (Maulsby, Kittlitz, & Witten, 1989).

Metamouse demonstrates that it is indeed possible for users to create graphical procedures by direct manipulation. Applications range from producing complex, repetitive drawings, through constructively specifying figures governed by graphical constraint, to generating simple animated algorithms for tasks such as sorting. Metamouse reveals its predictions as soon as it can. This has three advantages. First, users reap early benefits when performing repetitive operations. Second, they can correct errors as soon as they occur. Third, they develop confidence in their programs without ever viewing any kind of listing. The principal shortcomings of the current system are its limited repertoire of graphical objects and transformations, the lack of a formal underpinning for the constraint model, and our limited experience of how users react to the new experience of working with Metamouse.

Acknowledgments. This study was supported by the Natural Sciences and Engineering Research Council of Canada and by Apple Computer, Inc. We gratefully acknowledge the key role Bruce MacDonald has played in helping us to develop our ideas and the stimulating research environment provided by the Knowledge Science Lab at the University of Calgary. We have benefited from contributions to this work

by Fritz Huber, Greg James, and Antonija Mitrovic. Many thanks are due to Allen Cypher, Rosanna Heise, Ted Kaehler, Alan Kay, David Kosbie, and Dave Smith for their insightful comments. Finally, we thank the editor and reviewers for their discriminating advice.

REFERENCES

- Abbott, E. A. (1884). *Flatland—A romance of many dimensions*. New York: Signet.
- Andreae, P. M. (1985). *Justified generalization: Acquiring procedures from examples*. Unpublished PhD dissertation, MIT, Department of Electrical Engineering and Computer Science, Boston.
- Angluin, D., & Smith, C. H. (1983). Inductive inference: Theory and methods. *Computing Surveys*, 15, 237-269.
- Bier, E. A., & Stone, M. C. (1986). Snap-dragging. *Proceedings of ACM SIGGRAPH*, 233-240. Dallas: ACM.
- Borning, A. (1981). The programming language aspects of ThingLab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3, 353-387.
- Borning, A. (1986). Defining constraints graphically. *Proceedings of ACM SIGCHI*, 137-143. Boston: ACM.
- Cutter, M., Halpern, B., & Spiegel, J. (1987). *MacDraw* [Computer program]. Cupertino, CA: Apple Computer Inc.
- Dennett, D. C. (1987). *The intentional stance*. Cambridge, MA: MIT Press.
- Ellman, T. (1989). Explanation-based learning: A survey of programs and perspectives. *Computing Surveys*, 21, 163-221.
- Fikes, R. E., & Nilsson, N. J. (1971). STRIPS—A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-288.
- Foley, J. D., & Van Dam, A. (1982). *Fundamentals of interactive computer graphics*. Reading, MA: Addison-Wesley.
- Fuller, N., & Prusinkiewicz, P. (1988). Geometric modeling with Euclidean constructions. *Proceedings of Computer Graphics International*, 379-391. Geneva: Springer-Verlag.
- Halbert, D. (1984). *Programming by example* (Research Rep. No. OSD-T8402). Palo Alto, CA: Xerox PARC.
- Heise, R. (1989). *Demonstration instead of programming*. Unpublished MSc thesis, University of Calgary, Department of Computer Science, Calgary, Alberta, Canada.
- Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C., Beard, M., & Mackey, K. (1989, September). The Xerox Star: A retrospective. *IEEE Computer*, pp. 11-29.
- Kin, N., Noma, T., & Kunii, T. L. (1989). Picture Editor: A 2D picture editing system based on geometric constructions and constraints. *Proceedings of Computer Graphics International*, 193-207. Leeds, England: Springer-Verlag.
- Kurlander, D., & Bier, E. A. (1988). Graphical search and replace. *Proceedings of ACM SIGGRAPH*, 113-120. Atlanta: ACM.
- MacDonald, B. A., & Witten, I. H. (1987). Programming computer controlled

- systems by non-experts. *Proceedings of the IEEE SMC Annual Conference*, 432-437. Alexandria, VA: IEEE.
- Maulsby, D. L., James, G. A., & Witten, I. H. (1989). Acquiring graphical know-how: An apprenticeship model. *Proceedings of European Knowledge Acquisition Workshop*, 406-419. Paris: Tirages-Express.
- Maulsby, D. L., Kittlitz, K. A., & Witten, I. H. (1989). Metamouse: Specifying graphical procedures by example. *Proceedings of ACM SIGGRAPH*, 127-136. Boston.
- Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18, 203-226.
- Myers, B. A. (1988). *Creating user interfaces by demonstration*. San Diego: Academic.
- Noma, T., Kunii, T. L., Kin, N., Enomoto, H., Aso, E., & Yamamoto, T. Y. (1988). Drawing input through geometrical constructions: Specification and applications. *Proceedings of Computer Graphics International*, 403-415. Geneva: Springer-Verlag.
- Pence, J., & Wakefield, C. (1988). *Tempo II* [Computer program]. Boulder, CO: Affinity Microsystems.
- Preparata, F. P., & Shamos, M. I. (1985). *Computational geometry*. New York: Springer-Verlag.
- Rich, C., & Waters, R. (1988, November). The programmer's apprentice: A research overview. *IEEE Computer*, pp. 11-25.
- Smith, D. C., Irby, C., Kimball, R., Verplank, W., & Harslem, E. (1982, April). Designing the Star user interface. *Byte*, pp. 242-282.
- Sutherland, I. E. (1963). Sketchpad: A man-machine graphical communication system. *Proceedings of the AFIPS Spring Joint Computer Conference*, 23, 329-346.
- Sutherland, I. E., & Hodgman, G. W. (1974). Reentrant polygon clipping. *Communications of the ACM*, 17, 32-42.
- Van Lehn, K. (1983). *Felicity conditions for human skill acquisition: Validating an AI-based theory* (Research Rep. No. CIS-21). Palo Alto, CA: Xerox PARC.
- van Sommers, P. (1984). *Drawing and cognition*. Cambridge, England: Cambridge University Press.
- White, R. M. (1988). Applying direct manipulation to geometric construction systems. *Proceedings of Computer Graphics International*, 446-455. Geneva: Springer-Verlag.
- Williams, G. (1984, February). The Apple Macintosh computer. *Byte*, pp. 30-54.
- Witten, I. H., & MacDonald, B. A. (1988). Using concept learning for knowledge acquisition. *International Journal of Man-Machine Studies*, 29, 171-196.

HCI Editorial Record. First manuscript received May 31, 1990. Revision received March 18, 1991. Accepted by Brad Myers. Final manuscript received July 10, 1991. — Editor

Fitts' Law as a Research and Design Tool in Human-Computer Interaction

I. Scott MacKenzie
University of Toronto

ABSTRACT

According to Fitts' law, human movement can be modeled by analogy to the transmission of information. Fitts' popular model has been widely adopted in numerous research areas, including kinematics, human factors, and (recently) human-computer interaction (HCI). The present study provides a historical and theoretical context for the model, including an analysis of problems that have emerged through the systematic deviation of observations from predictions. Refinements to the model are described, including a formulation for the index of task difficulty that is claimed to be more theoretically sound than Fitts' original formulation. The model's utility in predicting the time to position a cursor and select a target is explored through a review of six Fitts' law studies employing devices such as the mouse, trackball, joystick, touchpad, helmet-mounted sight, and eye tracker. An analysis of the performance measures reveals tremendous inconsistencies, making across-study comparisons difficult. Sources of experimental variation are identified to reconcile these differences.

Author's present address: I. Scott MacKenzie, Department of Computing and Information Science, University of Guelph, Guelph, Ontario N1G 2W1, Canada.

CONTENTS

1. INTRODUCTION
 2. SUMMARY OF FITTS' LAW
 - 2.1. Information Theory Foundation
 - 2.2. Equation by Parts
 - 2.3. Physical Interpretation
 - 2.4. Derivation From a Theory of Movement
 3. DETAILED ANALYSIS
 - 3.1. The Original Experiments
 - 3.2. Problems Emerge
 - 3.3. Variations on Fitts' Law
 - 3.4. Effective Target Width
 - 3.5. Reanalysis of Fitts' Data
 - 3.6. Effective Target Amplitude
 - 3.7. Targets and Angles
 4. COMPETING MODELS
 - 4.1. The Linear Speed-Accuracy Tradeoff
 - 4.2. Power Functions
 5. APPLICATIONS OF FITTS' LAW
 - 5.1. The Generality of Fitts' Law
 - 5.2. Review of Six Studies
 - Card, English, and Burr (1978)
 - Drury (1975)
 - Epps (1986)
 - Jagacinski and Monk (1985)
 - Kantowitz and Elvers (1988)
 - Ware and Mikaelian (1987)
 - 5.3. Across-Study Comparison of Performance Measures
 - 5.4. Sources of Variation
 - Device Differences
 - Task Differences
 - Selection Technique
 - Range of Conditions and Choice of Model
 - Approach Angle and Target Width
 - Error Handling
 - Learning Effects
 - 5.5. Summary
 6. CONCLUSIONS
-

1. INTRODUCTION

Fitts' law is a model of human psychomotor behavior derived from Shannon's Theorem 17, a fundamental theorem of communication systems (Fitts, 1954; Shannon & Weaver, 1949). The realization of movement in Fitts'

model is analogous to the transmission of information. Movements are assigned indices of difficulty (in units of *bits*), and in carrying out a movement task the human motor system is said to transmit so many "bits of information." If the number of bits is divided by the time to move, then a rate of transmission in "bits per second" can be ascertained.

In the decades since Fitts' original publication, his relationship, or law, has proven one of the most robust, highly cited, and widely adopted models to emerge from experimental psychology. Psychomotor studies in diverse settings—from under a microscope to under water—have consistently shown high correlations between Fitts' index of difficulty and the time to complete a movement task. Kinematics and human factors are two fields particularly rich in investigations of human performance using Fitts' analogy.

In the relatively new discipline of HCI, there is also an interest in the mathematical modeling and prediction of human performance using an information-processing model. The starting point for Fitts' law research in HCI is the work of Card, English, and Burr (1978). In comparing four devices for selecting text on a CRT display, the model provided good performance predictions for the joystick and mouse. More than 80% of the variation in movement time was accounted for by the regression equations. In the subsequent Keystroke-Level Model for predicting user performance times (Card, Moran, & Newell, 1980), Fitts' law was cited as an appropriate tool for predicting pointing time but was omitted from the model in lieu of a constant. The value $t_p = 1.10$ s was derived from the Fitts' law prediction equation in Card et al. (1978) and served as a good approximation for pointing time over the range of conditions employed. Similarly, the Model Human Processor of Card, Moran, and Newell (1983, p. 26) comprises nine principles of operation. These have been the focus of a substantial body of empirical research leading to a psychological model of the human as an information processor. As the performance model for the human motor processor, Fitts' law, Principle P5, plays a prominent role in the Model Human Processor.

The need for a reliable prediction model of movement time in computer input tasks is stronger today than ever before. Bit-mapped graphic displays have all but replaced character-mapped displays, and office and desktop metaphors are gaining in popularity over menus and command lines. Today's user interfaces often supplant cursor keys and function keys with mice and pull-down menus. As the man-machine link gets more "direct," speed-accuracy models for human movement become ever closer to actions in human-computer dialogues. Design models, such as the Keystroke-Level Model, need to express the current range of movement activities in computer input tasks. Fitts' law can fill that need.

This study endeavors to critically assess the current state of Fitts' law and to suggest ways in which future research and design may benefit from a

rigorous and slightly corrected adaptation of this powerful model. Newell and Card (1985) expanded on the role for theoretical models in the design of human-computer interfaces:

Another way [for theory to participate] is through explicit computer program tools for the design. The theory is embodied in the tool itself, so that when the designer uses the tool, the effect of the theory comes through, whether he or she understands the theory or not. (p. 223)

Psychological theories and experiments, such as Fitts' index of difficulty . . . can shape the way a designer thinks about a problem. Analyses of the key constraints of a problem can point the way to fertile parts of the design space. Providing tools for thought is a more effective way of getting human engineering into the interface than running experiment comparisons between alternative designs. (p. 238)

Certainly though, conducting empirical experiments to validate models is the starting point. Putting the theory into tools comes later. When properly applied and integrated into tools, however, theories may indeed elicit new ways of thinking for designers.

The theory underlying Fitts' relationship is sufficiently complex, and the ideas presented here are so subtle that a thorough analysis of the model is warranted before examining its applications. We begin with an overview of the most common interpretation of the law and then review the original experiments. Unlike many models that through statistical techniques yield parameters and constants void of physical interpretation, a key feature of Fitts' law is the correspondence to physical properties underlying movement tasks. An interpretation is offered for each term in the equation.

In the wake of the consistent departure of observations from predictions, many follow-up studies questioned the validity of the model. An analysis of Fitts' original data highlights these problems, with a correction offered that brings the model closer to the information theorem on which it is based. To complete the picture, several competing models are presented and compared with Fitts' law. Other research revealing the generality of the model in diverse and unusual settings is cited.

With this foundation, we undertake the task of connecting the theory to practical problems in HCI. Six studies are surveyed where Fitts' law was applied to input tasks using devices such as the mouse, trackball, joystick, touchpad, helmet-mounted sight, and eye tracker. Unfortunately, the results vary considerably, making across-study comparisons difficult. It is shown that task differences, selection techniques, range of conditions employed, and dealing with response variability (viz., errors) are among the major sources of experimental variation. An understanding of these increases the potential for

valid across-study comparisons and allows designers to benefit from a substantial body of existing Fitts' law research.

2. SUMMARY OF FITTS' LAW

Following the work of Shannon, Wiener, and other information theorists in the 1940s, information models of psychological processes emerged with great fanfare in the 1950s (e.g., see Miller, 1953; Pierce, 1961, chap. 12). The terms *probability*, *redundancy*, *bits*, *noise*, and *channels* entered the vocabulary of experimental psychologists as they explored the latest technique of measuring and modeling human behavior. Two surviving models are the Hick-Hyman law for choice reaction time (Hick, 1952; Hyman, 1953) and Fitts' law for the channel capacity of the human motor system (Fitts, 1954; Fitts & Peterson, 1964).

2.1. Information Theory Foundation

Fitts' idea was novel for two reasons: First, it suggested that the difficulty of a task could be measured using the information metric bits; second, it introduced the idea that, in carrying out a movement task, information is transmitted through a channel—a human channel. With respect to electronic communications systems, the concept of a channel is straightforward: A signal is transmitted through a nonideal medium (such as copper or air) and is perturbed by noise. The effect of the noise is to limit the information capacity of the channel below its theoretical maximum. Shannon's Theorem 17 expresses the effective information capacity C (in bits/s) of a communications channel of bandwidth B (in 1/s or Hz) as:

$$C = B \log_2 \frac{S + N}{N}, \quad (1)$$

where S is the signal power and N is the noise power (Shannon & Weaver, 1949, pp. 100–103).

The notions of *channel* and *channel capacity* are not as straightforward in the domain of human performance. The problem lies in the measurement of human channel capacity. Although electronic communications systems transmit information with specific and optimized codes, this is not true of human channels. Human coding is ill-defined, personal, and often irrational or unpredictable. Optimization is dynamic and intuitive. Cognitive strategies emerge in everyday tasks through *chunking*, which is analogous to *coding* in information theory—the mapping of a diverse pattern (or complex behavior) into a simple pattern (or behavior). Neuromuscular coding emerges through the interaction of nerve, muscle, and limb groups during the acquisition and repetition of skilled behavior. Difficulties in identifying and measuring

cognitive and neuromuscular factors confound the measurement of the human channel capacity, causing tremendous variation to surface in different experiments seeking to investigate similar processes.

2.2. Equation by Parts

Fitts sought to establish the information capacity of the human motor system. This capacity, which he called the index of performance (*IP*), is analogous to channel capacity (*C*) in Shannon's theorem. *IP* is calculated by dividing a motor task's index of difficulty (*ID*) by the movement time (*MT*) to complete a motor task. Thus,

$$IP = ID/MT. \quad (2)$$

Equation 2 parallels Equation 1 directly, with *IP* matching *C* (in bits/s), *ID* matching the log term (in bits), and *MT* matching $1/B$ (in seconds).

Fitts claimed that electronic signals are analogous to movement distances or amplitudes (*A*) and that noise is analogous to the tolerance or width (*W*) of the region within which a move terminates. Loosely based on Shannon's logarithmic expression, the following was offered as the index of difficulty for a motor task:

$$ID = \log_2(2A/W). \quad (3)$$

Because *A* and *W* are both measures of distance, the ratio within the logarithm is without units. The use of bits as the unit of task difficulty stems from the somewhat arbitrary choice of base 2 for the logarithm. (Had base 10 been used, the units would be digits.)

A useful variation of Equation 2 places *MT* on the left as the predicted variable:

$$MT = ID/IP. \quad (4)$$

This relationship is tested by devising a series of movement tasks with *ID* (viz., *A* and *W*) as the independent variable and *MT* as the dependent variable. In an experimental setting, subjects move to and acquire targets of width *W* at a distance *A* as quickly and accurately as possible. (*Accurate*, for the moment, implies a small but consistent error rate.) Several levels are provided for each of *A* and *W*, yielding a range of task difficulties.

The index of performance *IP* can be calculated directly using Equation 2 by dividing a task's index of difficulty by the observed movement time (averaged

over a block of trials), or it can be determined by regressing MT on ID . In the latter case, the regression line equation is of the form:

$$MT = a + b ID, \quad (5)$$

where a and b are regression coefficients. The reciprocal of the slope coefficient, $1/b$, corresponds to IP in Equation 4.¹ The usual form of Fitts' law is Equation 5 expanded as follows:

$$MT = a + b \log_2(2A/W). \quad (6)$$

The factor 2 in the logarithm was added by Fitts as an arbitrary adjustment to ensure that ID was greater than zero for the range of experimental conditions employed in his experiments (Fitts, 1954, p. 388). The 2 increases the index of difficulty by 1 bit for each task condition but does not affect the MT - ID correlation or the slope of the regression line.²

2.3. Physical Interpretation

A common experimental method for model building is the stepwise entering of parameters into a regression analysis on an ad hoc basis. Although the goal of accounting for variation in observed behavior is met, there is a cost:

over-parameterization . . . presents difficulties in terms of interpreting the meaning of parameter variations. This subverts some of the purposes of modeling, namely, providing succinct explanations of data and providing assistance in designing experiments. (Rouse, 1980, p. 6)

This is not the case with Fitts' law. A key feature of the model is the physical interpretation afforded by the parameters and empirically determined constants in the prediction equation.

As measures of magnitude, target amplitude and target width have straightforward interpretations: Big targets at close range are acquired faster than small targets at a distance. But the model predicts movement time as a function of a task's index of difficulty—the logarithm of the ratio of target amplitude to target width. This is a very convenient relationship. From Equation 3, task difficulty (ID) increases by 1 bit if target distance is doubled

¹ Throughout this article, the following units are consistently used: bits/s for IP , ms/bit for b , and ms for MT and a .

² The 2 may also be explained by expressing the log term as $\log_2(A/\frac{1}{2}W)$, where A is the distance moved and $\frac{1}{2}W$ is the size of the error band on each side of target center.

or if the size is halved. Thus, *ID* provides a useful, single measure of the combined effect of two physical properties of movement tasks.

The intercept (*a*) and slope (*b*) coefficients in Equation 6 are empirically determined constants. Ideally the intercept is zero, suggesting that a task of zero difficulty takes 0 s; however, linear regression usually produces a nonzero value. Although the magnitude of the intercept is viewed by some as an indication of the model's accuracy, a substantial positive intercept indicates the presence of an additive factor unrelated to the index of difficulty. Target acquisition tasks on computers are particularly sensitive to additive factors. The *select* operation, which typically follows pointing, may entail a button push, the application of pressure, dwell time, and so on. These responses should have an additive effect, contributing to the intercept of the regression line but not to the slope.

Fitts' index of performance is the reciprocal of the regression line slope and carries the units bits per second. In executing a movement task, *ID* is the number of bits of information transmitted, and *IP* is the rate of transmission. Although it is glossed over in many accounts of the model, Fitts' thesis was that *IP* is constant across a range of values for *ID*. It follows that the relationship between *MT* and *ID* is linear. His experiments provided strong evidence to support this claim, as has a large body of subsequent research.

Many studies have sought to establish the human rate of information processing in diverse settings. Langolf, Chaffin, and Foulke (1976) tested different limb groups and found that *IP* decreased as the limb changed from the finger to the wrist to the arm. This implies that large, cumbersome limbs are more sensitive to changes in *ID* than small dexterous limbs. There is a vital role for this sort of knowledge in the design of high-performance man-machine interfaces.

2.4. Derivation From a Theory of Movement

Fitts deduced his model by analogy. Trying to explain why the analogy works so well and to justify the model from a low-level account of the underlying phenomena has challenged psychomotor researchers ever since. Devising a theory and providing a derivation is not so simple, however. Pew and Baron (1983, p. 664) claimed that:

There is no useful distinction between models and theories. We assert that there is a continuum along which models vary that has loose verbal analogy and metaphor at one end and closed-form mathematical equations at the other, and that most models lie somewhere in-between.

Fitts' law may be placed in this continuum. As a mathematical expression, it emerged from the rigors of probability theory, yet when applied to

psychomotor behavior it becomes a metaphor. Derivations of the law, therefore, must build on assumptions—assumptions on the perceptual, psychological, and physiological processes underlying human movement. A derivation explains a model well if it requires only a few simple assumptions that can be validated in the laboratory.

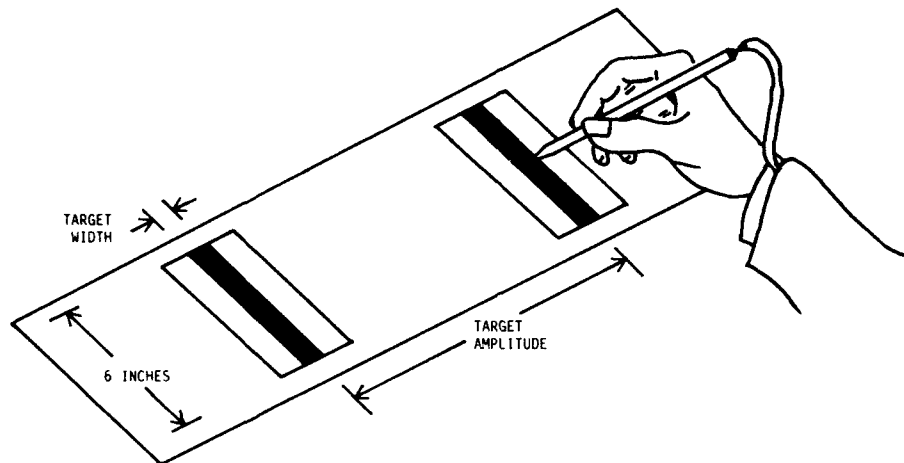
The most accepted derivation originates from the *deterministic iterative-corrections model*, originally offered by Crossman and Goodeve (1963/1983) and developed subsequently by others (e.g., Keele, 1968; Langolf et al., 1976). The derivation builds on the underlying assumption that a complete move is realized through iterations of feedback-guided corrective submovements. A move is assumed to take n submovements, each taking a constant time of t seconds to complete. It follows that the time to complete a move is nt seconds. A constant of proportionality (p) is introduced such that for each submovement the distance covered is $1 - p$ times the distance remaining.

Based on these assumptions, the derivation proceeds as follows. After the first submovement in a move of total distance A , the distance moved is $(1 - p)A$ and the distance remaining is pA . After the second submovement, the distance moved is $(1 - p)pA$ and the distance remaining is ppA or p^2A . After n submovements, the distance remaining is p^nA . Completing a move within the target implies that the distance remaining is $\leq \frac{1}{2}W$. Setting $p^nA = \frac{1}{2}W$ and solving for n yields $b' \log_2(2A/W)$, where b' is the constant $-1/\log_2 p$ (which must be > 1 because $0 < p < 1$). The time to complete a move is $MT = nt = b \log_2(2A/W)$, where b is the positive constant $b't$. This is the same as Fitts' law (Equation 6) except the intercept, a , is missing. The intercept may be accounted for by noting that the first move should take less than t (by a constant a) because the time to decide how far to move initially occurs before a move begins (Keele, 1968).

One way of testing the derivation is to fix values for t and p , and calculate b . Estimates for t , the time to process visual feedback, are in the range of 135 ms to 290 ms (Beggs & Howarth, 1970; Carlton, 1981; Crossman & Goodeve, 1963/1983; Keele & Posner, 1968). The proportional error constant, p , is between .04 and .07 (Langolf et al., 1976; Meyer, Abrams, Kornblum, Wright, & Smith, 1988; Pew, 1974; Schmidt, 1988, p. 275; Vince, 1948). Using $t = 290$ ms and $p = .07$ yields $b = -t/\log_2 p = 75.6$ ms/bit or $IP = 1/b = 13.2$ bits/s, a value close to that found by Fitts (Fitts & Peterson, 1964).

Despite the appealing simplicity of the deterministic iterative-corrections model, the underlying assumptions are suspect. Langolf et al. (1976) found that some movements have only one correction despite the prediction of several corrective submovements when A/W is appreciable. Jagacinski, Repperger, Moran, Ward, and Glass (1980) questioned the hypothesis of constant-duration submovements, having found considerable variation in the duration of the initial submovement. Also, the model is completely deterministic and cannot explain why subjects sometime miss a target and commit an error (Meyer, Smith, Kornblum, Abrams, & Wright, 1990).

Figure 1. Fitts' reciprocal tapping paradigm (after Fitts, 1954).



So, despite being robust and highly replicable, Fitts' law remains an analogy waiting for a theory. Providing a reasonable account of the law through a theory of human movement—rather than a theory of information—remains a research goal.

3. DETAILED ANALYSIS

Fitts' original experiments provide the basis for a detailed examination of the model's utility, shortcoming and universality. Following an analysis of Fitts' work, problems and weaknesses in the model are examined in view of a substantial body of follow-up research.

3.1. The Original Experiments

The original investigation (Fitts, 1954) involved four experiments: two reciprocal tapping tasks (1-oz stylus and 1-lb stylus), a disc transfer task, and a pin transfer task. In the tapping experiments, subjects moved a stylus back and forth between two plates as quickly as possible and tapped the plates at their centers (see Figure 1). This experimental arrangement is commonly called the "Fitts' paradigm."

Because summary data were published in Fitts' original report, and because this work is so vital to our investigation, these experiments are analyzed in detail to develop (and correct) some of the concepts in the information-processing analogy. Figure 2 reproduces the data from the 1-oz tapping experiment, with one column of additional data that is discussed soon.

Target width and target amplitude varied across four levels, resulting in

Figure 2. Data from Fitts' (1954) reciprocal task experiment with 1-oz stylus. An extra column shows the effective target width (W_e) after adjusting W for the percentage errors.

W (in.)	W_e^a (in.)	A (in.)	ID (Bits)	MT (ms)	Errors (%)	IP^b (Bits/s)
0.25	0.243	2	4	392	3.35	10.20
0.25	0.244	4	5	484	3.41	10.33
0.25	0.235	8	6	580	2.78	10.34
0.25	0.247	16	7	731	3.65	9.58
0.50	0.444	2	3	281	1.99	10.68
0.50	0.468	4	4	372	2.72	10.75
0.50	0.446	8	5	469	2.05	10.66
0.50	0.468	16	6	595	2.73	10.08
1.00	0.725	2	2	212	0.44	9.43
1.00	0.812	4	3	260	1.09	11.54
1.00	0.914	8	4	357	2.38	11.20
1.00	0.832	16	5	481	1.30	10.40
2.00	1.020	2	1	180	0.00	5.56
2.00	1.233	4	2	203	0.08	9.85
2.00	1.576	8	3	279	0.87	10.75
2.00	1.519	16	4	388	0.65	10.31
M				392	1.84	10.10
SD				157	1.22	1.33

^aData added (see text). ^b $IP = ID / MT$.

ID s of 1 to 7 bits. Mean MT s ranged from 180 ms to 731 ms, with each mean derived from more than 600 observations. In assuming an intercept of zero (see Equation 4), Fitts calculated IP directly by dividing ID by MT for each experimental condition. A quick glance at Figure 2 shows the strong evidence for the thesis that the rate of information processing is constant across a range of task difficulties. The mean value of $IP = 10.10$ bits/s ($SD = 1.33$ bits/s) is purportedly the information-processing rate of the human motor system.

Although Fitts did not perform correlation or regression analyses on his 1954 data, others have. Correlating MT with ID yields $r = .9831$ ($p < .001$). It is noteworthy of the model in general that correlations above .9000 consistently emerge. Regressing MT on ID results in the following prediction equation for MT (in ms):

$$MT = 12.8 + 94.7 ID. \quad (7)$$

Calculating IP from the reciprocal of the slope yields an information-processing rate of 10.6 bits/s. This rate is slightly higher than that obtained through direct calculation because it is derived from a least-squares regression equation with a positive intercept. When IP is calculated directly, the linear

relationship takes on an intercept of zero. A positive intercept reduces the slope of the line, thus increasing *IP*. Although some researchers cite values of *IP* calculated directly (notably Fitts, 1954), most use the statistical technique of linear regression and provide a value for *IP* (the reciprocal of the slope) and an intercept. See Sugden (1980) or Salmoni and McIlwain (1979) for further discussions on the merits of each technique of calculating *IP*.

3.2. Problems Emerge

Despite the high correlation between *ID* and the observed mean *MT*, problems have been noted. Scatter plots often reveal an upward curvature of *MT* away from the regression line for low values of *ID* (see Figure 3). This systematic departure of observations from predictions was first pointed out by Crossman in 1957 (Welford, 1960) and has been observed in other studies since (Buck, 1986; Crossman & Goodeve, 1963/1983; Drury, 1975; Klapp, 1975; Langolf et al., 1976; Meyer et al., 1988; Meyer et al., 1990; Wallace, Newell, & Wade, 1978).

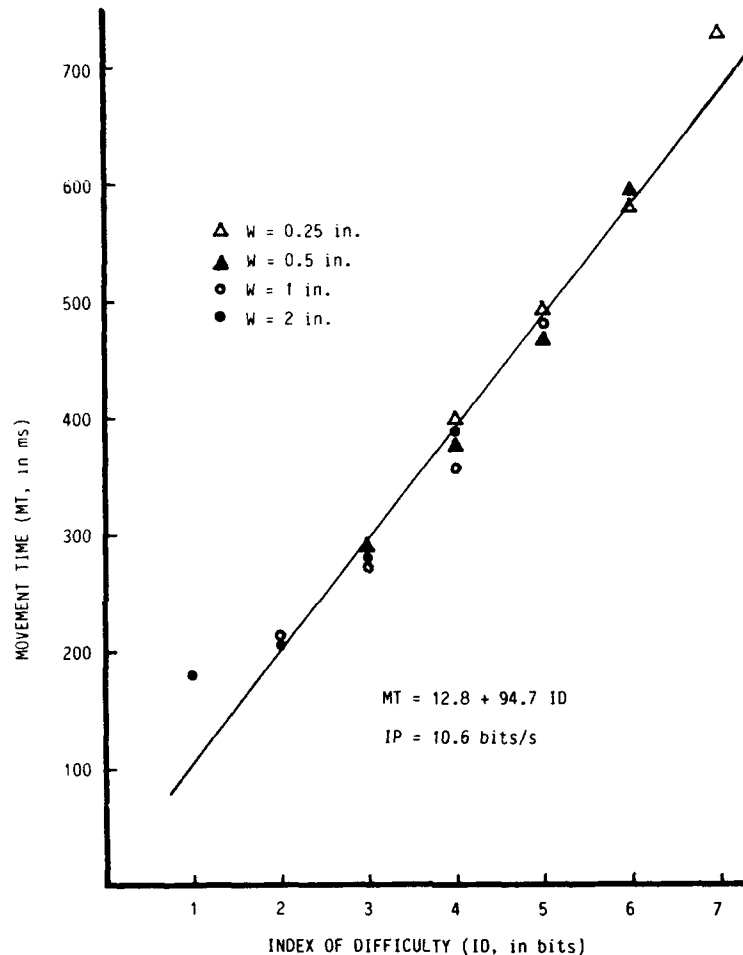
The failure of the model when *ID* is small is also evident in Figure 2. The *IP* rating of 5.56 bits/s for *ID* = 1 bit is 3.4 *SDs* from the mean value of 10.10 bits/s.

Another problem stems from the relative contributions of *A* and *W* in the prediction equation. Accordingly, the effect should be equal but inverse. A doubling of the target amplitude adds 1 bit to the index of difficulty and increases the predicted movement time. The same effect is predicted from Equation 6 if target width is halved. In an analysis of Fitts' (1954) four experiments, M. R. Sheridan (1979) showed that reductions in target width cause a disproportionate increase in movement time when compared to similar increases in target amplitude. Others have also independently noted this disparity (Keele, 1973, p. 112; Meyer et al., 1988; Welford, Norris, & Shock, 1969). It is also evident in the scatter plots in some reports, although not noted by the investigators (Buck, 1986; Jagacinski & Monk, 1985; Jagacinski, Repperger, Ward, & Moran, 1980).

An error-rate analysis may also reveal the inequitable contributions of *A* and *W*. Wade, Newell, and Wallace (1978) found a significant main effect between error rate and target width, $F(2, 40) = 16.60$, $p < .01$, with errors increasing as target width decreased but no main effect between error rate and target amplitude. A similar observation was made by Card et al. (1978).

By no means is there unanimity on the point just raised. When *ID* is less than around 3 bits, movements are brief and feedback mechanisms yield to impulse-driven ballistic control. The disparity may be just the opposite under these conditions. Gan and Hoffmann (1988) found that when *ID* is small *MT* is strongly dependent on movement amplitudes, with no significant effects from target width.

Figure 3. Scatter plot of movement time versus index of difficulty. Sixteen combinations of A and W were employed with ID s ranging from 1 to 7 bits (after Fitts, 1954).



Fitts' analogy has proven itself in many settings, but, like all models, limitations and inaccuracies emerge under extremes of conditions or when the grain of analysis is fine.

3.3. Variations on Fitts' Law

In an effort to improve the data-to-model fit, numerous researchers have proposed variations on Fitts' relationship or have introduced new models derived from different principles. Welford's (1960; 1968, p. 147) variation is the most widely adopted, and it commonly appears in two forms:

$$MT = a + b \log_2(A/W + 0.5) \quad (8)$$

or

$$MT = a + b \log_2 \frac{A + 0.5W}{W}. \quad (9)$$

The latter form is strikingly similar to Shannon's original theorem (cf. Equation 1). Many researchers, including Fitts, have reported higher correlations between *MT* and *ID* using Welford's formulation (Beggs, Graham, Monk, Shaw, & Howarth, 1972; Drury, 1975; Fitts & Peterson, 1964; B. A. Kerr & Langolf, 1977; Knight & Dagnall, 1967; Kvålseth, 1980). Although Fitts' original formulation (Equation 6) is still the most frequently used, many researchers (most notably in the present context, Card et al., 1978) prefer Equation 8.

Recently it was shown that Fitts deduced his relationship citing an approximation of Shannon's theorem originally introduced with the caution that it is useful only if the signal-to-noise ratio is large (Fitts, 1954, p. 388; Goldman, 1953, p. 157; MacKenzie, 1989). The signal-to-noise ratio in Shannon's theorem corresponds to the ratio of target amplitude to target width in Fitts' analogy. As evident in Figure 2, Fitts' experiments extended the *A:W* ratio as low as 1:1! The variation of Fitts' law suggested by direct analogy with Shannon's information theorem is:

$$MT = a + b \log_2(A/W + 1) \quad (10)$$

or the alternate form

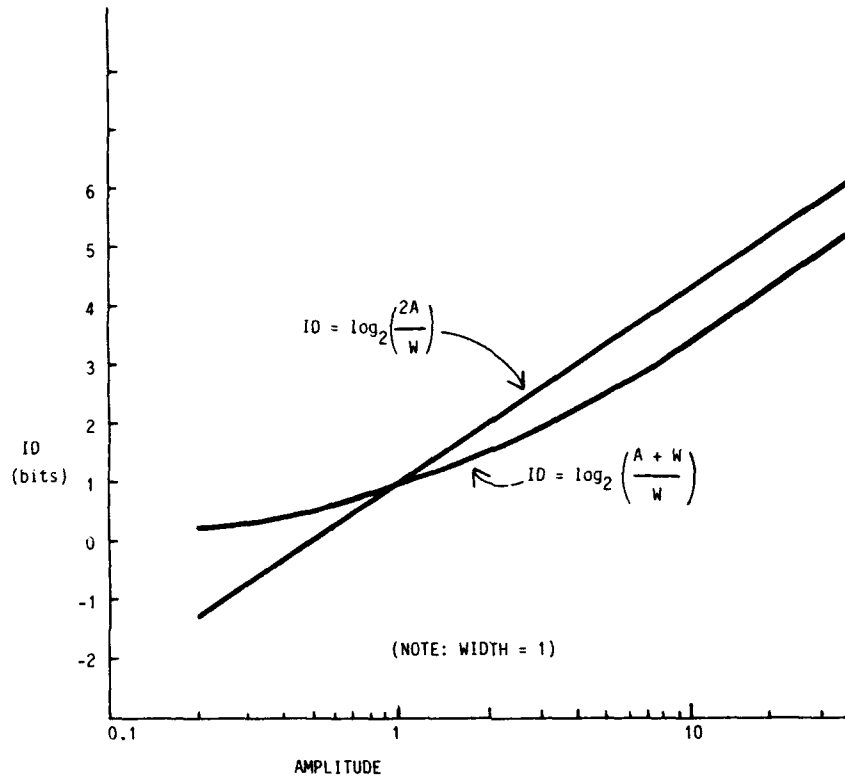
$$MT = a + b \log_2 \frac{A + W}{W}. \quad (11)$$

The difference between Equation 10 and Equation 6 (Fitts' law) is illustrated by comparing changes in the logarithm term (*ID*) as *A* approaches zero with *W* held constant (see Figure 4). It is noteworthy of Equation 10 that the logarithm cannot be negative.³

Obviously, a negative rating for task difficulty presents a serious theoretical problem. It is a minor consolation that this can only occur with Fitts' expression when the targets overlap, that is, when $A < W/2$. Although such conditions may seem unreasonable, the possibility has been investigated before (Schmidt, 1988, p. 271; Welford, 1968, p. 145) and can occur when output measures are adjusted to reflect the variance in subjects' responses (using a technique described shortly). Regardless, researchers have actually

³ Welford's formulation produces a similar curve to Equation 10 except that *ID* approaches -1 bit as *A* approaches zero.

Figure 4. Comparison of Fitts' index of difficulty and an *ID* based on Shannon's Theorem 17. *W* is held constant at 1 unit as *A* approaches zero.



reported on experimental conditions with a negative *ID* (e.g., Card et al., 1978; Crossman & Goodeve, 1963/1983; Gillan, Holden, Adam, Rudisill, & Magee, 1990; Ware & Mikaelian, 1987).

The practical consequences of using Equation 10 in lieu of Fitts' or Welford's equation are probably slight and are likely to surface only in experimental settings with *ID*s extending under approximately 3 bits, as suggested from Figure 4. Nevertheless, the theoretical implications of Equation 10 are considerable. First, the idea that similar changes in target amplitude and target width should effect a similar but inverse change in movement time as suggested in Equation 6 does not follow in Equation 10.

Also, the sound theoretical premise for Equation 10 casts doubt on the rationale for a popular and mathematically correct transformation of Fitts' law, which separates *A* and *W*:

$$MT = a + b_1 \log_2 A - b_2 \log_2 W. \quad (12)$$

Welford (1968, p. 156) suggested that $b_1 \log_2 A$ may correspond to an initial open-loop impulse toward a target and that $b_2 \log_2 W$ may correspond to a feedback-guided final adjustment as a move terminates. Numerous researchers have used or analyzed Equation 12 with good results (Bainbridge & Sanders, 1972; Gan & Hoffmann, 1988; Jagacinski, Repperger, Moran, Ward, & Glass, 1980; Jagacinski, Repperger, Ward, & Moran, 1980; Kay, 1960; R. Kerr, 1978; M. R. Sheridan, 1979; Welford et al., 1969; Zelaznik, Mone, McCabe, & Thaman, 1988). In multiple correlation analyses, Equation 12 always yields a higher R than the single factor r obtained using Equation 6 (because of the extra degree of freedom); however, the model ceases to have an information-theoretic premise because similar recasting is not possible using Equation 10, which directly mimics Shannon's original theorem. For example, from Equation 12, What is ID ? and What is IP ?

Finally, derivations of Fitts' law, such as that provided by Crossman and Goodeve (1963/1983), cannot accommodate Equation 10 without introducing further assumptions. Thus, the Shannon formulation addresses several theoretical issues and offers slightly better prediction power than Fitts' or Welford's formulation.

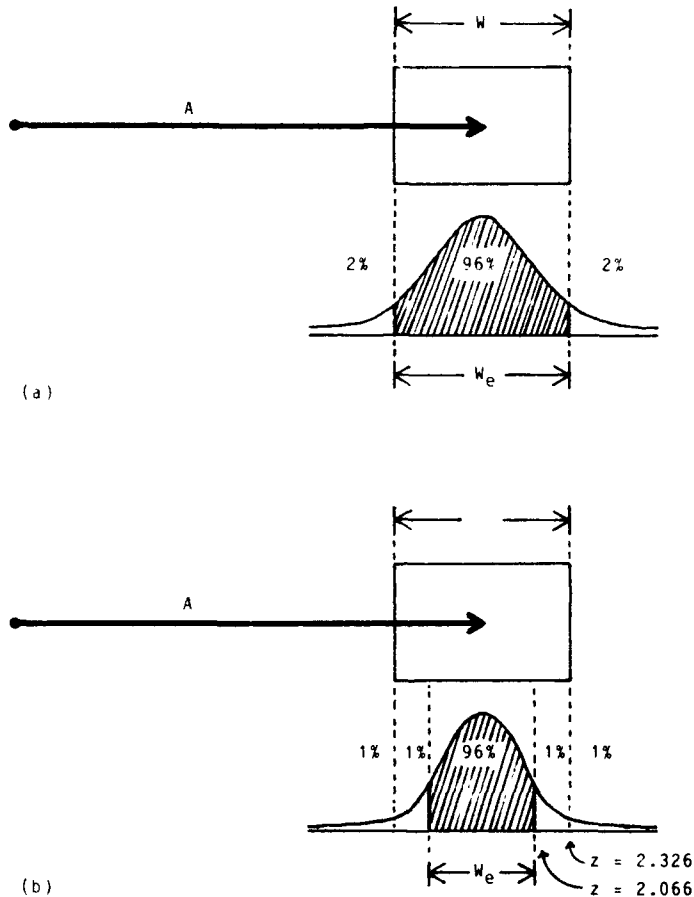
3.4. Effective Target Width

Of greater practical importance is a technique to adjust output measures to bring the model in line with the underlying principles. The technique calls for normalizing target width to reflect what a subject actually did (output condition), rather than what was expected (input condition). Thus, at the model-building stage, W becomes a dependent variable.

The output or "effective" target width (W_e) is derived from the distribution of "hits" (see Welford, 1968, pp. 147-148). This adjustment lies at the very heart of the information-theoretic metaphor—that movement amplitudes are analogous to "signals" and that endpoint variability (viz., target width) is analogous to "noise." In fact, the information theorem underlying Fitts' law assumes that the signal is "perturbed by white thermal noise" (Shannon & Weaver, 1949, p. 100). The analogous requirement in motor tasks is a Gaussian or normal distribution of hits—a property observed by numerous researchers (e.g., Crossman & Goodeve, 1963/1983; Fitts, 1954; Fitts & Radford, 1966; Welford, 1968, p. 154; Welford et al., 1969; Woodworth, 1899).

The experimental implication of normalizing output measures is illustrated as follows. The entropy, or information, in a normal distribution is, $\log_2 (\sqrt{2 \pi} \sigma) = \log_2 (4.133 \sigma)$, where σ is the standard deviation in the unit of measurement. Splitting the constant 4.133 into a pair of z scores for the unit-normal curve (i.e., $\sigma = 1$), one finds that 96% of the total area is

Figure 5. Method of adjusting target width based on the distribution of endpoint coordinates. When 4% errors occur, the effective target width, $W_e = W$. When less than 4% errors occur, $W_e < W$.



bounded by $-2.066 < z < +2.066$. In other words, a condition that target width is analogous to noise is that the distribution is normal with 96% of the hits falling within the target and 4% of the hits missing the target (see Figure 5a). When an error rate other than 4% is observed, target width can be adjusted to form the effective target width in keeping with the underlying theory. This is a crucial point that we dwell on in more detail later.

There are two methods for determining the effective target width. If the standard deviation of the endpoint coordinates is known, just multiply SD by 4.133 to get W_e . When percentage errors are known, the method is trickier and requires a table of z scores for areas under the unit-normal curve. The method is: If n percentage errors are observed for a particular $A-W$ condition,

determine z such that $\pm z$ contains $100 - n$ percent of the area under the unit-normal curve. Multiply W by $2.066/z$ to get W_e . For example, if 2% errors were recorded on a block of trials when tapping or selecting a 5-cm wide target, then $W_e = 2.066/z \times W = 2.066/2.326 \times 5 = 4.45$ cm (see Figure 5b).

Experiments following this approach may find the variation in IP reduced because, typically, subjects that take longer are more accurate and demonstrate less endpoint variability. Reduced variability decreases the effective target width and therefore increases the effective index of difficulty (see Equation 3). On the whole, an increase in MT is compensated for by an increase in the effective ID , and this tends to lessen the variability in IP (see Equation 2).

This technique is not new, yet it has been largely ignored in the published body of Fitts' law research that could have benefited from it.⁴ There are several possible reasons for the lack of use of this technique. First, the method is tricky and its derivation from information-theoretic principles is complicated (e.g., see Reza, 1961, pp. 278-282). Second, the endpoint coordinate must be recorded for each trial in order to calculate W_e from the standard deviation. This is feasible using a computer for data acquisition and statistical software for analysis, but manual measurement and data entry are extremely awkward.⁵

Inaccuracy may enter when adjustments use the percentage errors because the extreme tails of the unit-normal distribution are involved. It is necessary to use z scores with at least three decimal places of accuracy for the factoring ratio (which is multiplied by W to yield W_e). Manual look-up methods are prone to precision errors. Furthermore, some of the easier experimental conditions may have error rates too low to reveal the true distribution of hits. The technique cannot accommodate "perfect performance"! For example, as shown in Figure 2, 0.00% errors occurred when $A = W = 2$ in., which seems reasonable because the target edges were touching. This observation suggests a large adjustment because the distribution is very narrow (in comparison to the target width over which the hits should have been distributed—with 4% errors!). A pragmatic approach in this case is to assume an error rate of 0.0049% (which rounds to 0.00%) at worst and proceed to make the adjustment.

Introducing a post hoc adjustment on target width before the regression

⁴ The study by MacKenzie, Sellen, and Buxton (1991) is an exception. Fitts' law prediction equations were derived for the mouse, trackball, and tablet-with-stylus in both pointing and dragging tasks. The equations were derived using the Shannon formulation for ID and the effective target width, W_e .

⁵ Despite being more cumbersome, the standard deviation method is better than the discrete error method because more behavioral characteristics can be discerned, such as the predominance of overshoots versus undershoots or the presence of outliers.

analysis (or maintaining a consistent error rate of around 4%) is important to maintain the information-theoretic analogy. There is a tacit assumption in Fitts' law that subjects, although instructed to move "as quickly and accurately as possible," balance the demands of tasks to meet the spatial constraint that 96% of the hits fall within the target. When this condition is not met, an adjustment to target width should be introduced. Furthermore, if subjects slow down and place undue emphasis on accuracy, the task changes; the constraints become temporal, and the prediction power of the model falls off (Meyer et al., 1988). In summary, Fitts' law is a model for rapid, aimed movements, and the presence of a nominal yet consistent error rate in subjects' behavior is assumed and arguably vital.

3.5. Reanalysis of Fitts' Data

The technique for adjusting target width based on percentage errors was applied to the data in Fitts' tapping experiments in determining the effective target width. The adjusted values, W_e , are shown in Figure 2 for the 1-oz tapping experiment. The correlation between ID and MT for the first experiment using Fitts' model without the adjustments is high ($r = .9831$, $p < .001$), as previously noted, but higher when ID is recalculated using W_e ($r = .9904$, $p < .001$), and even higher using W_e and the Shannon formulation ($r = .9937$, $p < .001$).⁶ As evident in Figure 6, the trend is similar for the other experiments.⁷

A scatter plot of MT versus ID , where $ID = \log_2(A/W_e + 1)$ from Equation 10, shows a coalescing of points about the regression line (cf. Figures 3 and 7). Note that the range of ID s is narrower using adjusted measures. This is due to the 1-bit decrease when ID is greater than about 2 bits (see Figure 4) and the general increase in ID for "easy" tasks because of the narrow distribution of hits.

Although the regression equation obtained using Fitts' expression is noteworthy for providing the intercept closest to the origin for all four experiments (see Figure 6), the standard error is the highest for all experiments. In general, a large intercept is due to the presence of factors that are unaccounted for, such as a "button push" or other antagonistic muscle activity at the beginning or ending of a task (Keele, 1968; Meyer, Smith, & Wright,

⁶ A two-tailed t test shows that the difference between the Fitts and Shannon correlations ($r = .9904$ vs. $r = .9937$; both calculated using W_e) is statistically significant ($t = 2.20$, $df = 13$, $p < .05$; see MacKenzie, 1989). Welford's formulation consistently yields correlations between those using the Fitts and Shannon formulations.

⁷ The differences between the correlations in the disc and pin transfer experiments are not statistically significant; however, these experiments used ID s of 4 to 10 bits and 3 to 10 bits, respectively. As demonstrated in Figure 4, the Fitts and Shannon formulations differ significantly only when ID s extend under around 3 bits.

Figure 6. Reanalysis of data from Fitts' (1954) experiments. For each experimental condition, the trend is for the correlation to increase and the standard error to decrease when target width is adjusted for percentage errors and *ID* is calculated using the Shannon formulation. Target width could not be adjusted for the disc and pin transfer experiments because errors could not occur. Analysis was conducted using SPSS⁺ Release 3.1 (1990).

Model	Equation	Target Width	r^a	Regression Coefficient		IP (Bits/s)
				Intercept (SE) ^b	Slope (SE) ^b	
<i>1-oz Stylus</i>						
Fitts	6	Unadjusted (W)	.9831	12.8 (20.3)	94.7 (4.7)	10.6
Fitts	6	Adjusted (W_e)	.9904	- 73.2 (18.0)	108.9 (4.0)	9.2
Shannon	10	Adjusted (W_e)	.9937	- 31.4 (13.4)	122.0 (3.6)	8.2
<i>1-lb Stylus</i>						
Fitts	6	Unadjusted (W)	.9796	- 6.2 (24.7)	104.8 (5.7)	9.5
Fitts	6	Adjusted (W_e)	.9882	- 118.0 (22.8)	124.0 (5.1)	8.1
Shannon	10	Adjusted (W_e)	.9925	- 69.8 (16.6)	138.8 (4.5)	7.2
<i>Disc Transfer</i>						
Fitts	6	Unadjusted (W)	.9186	150.0 (74.6)	90.4 (10.4)	11.1
Shannon	10	Unadjusted (W)	.9195	223.5 (66.0)	92.6 (10.6)	10.8
<i>Pin Transfer</i>						
Fitts	6	Unadjusted (W)	.9432	22.3 (48.2)	86.1 (7.1)	11.6
Shannon	10	Unadjusted (W)	.9452	84.4 (42.4)	89.4 (7.3)	11.2

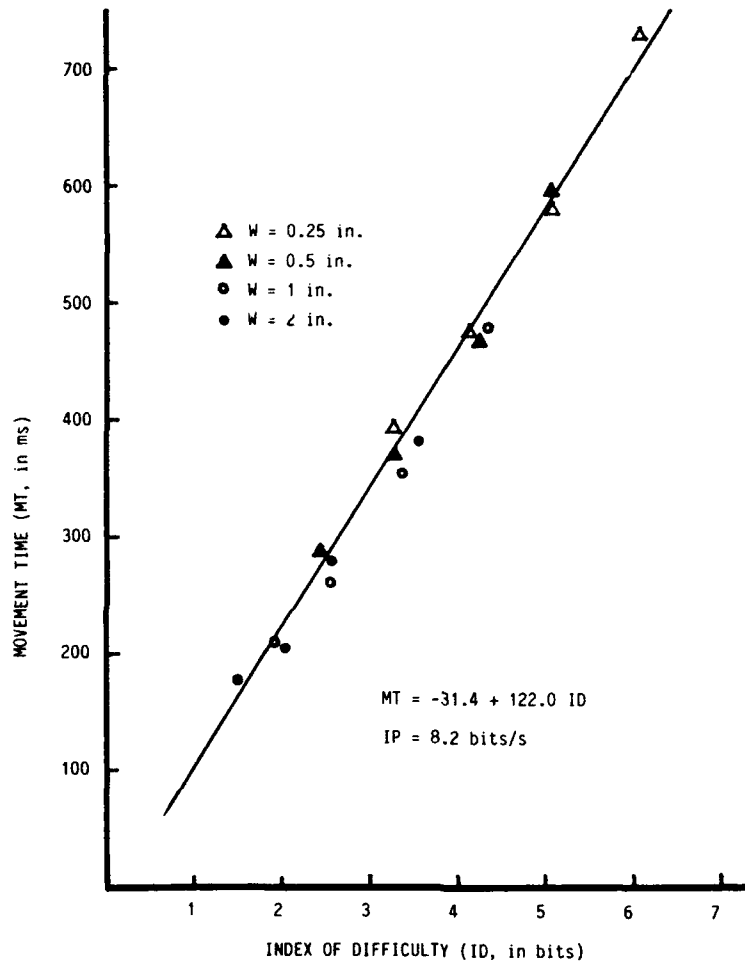
^a $p < .001$. ^bStandard error.

1982). In follow-up applications, a negative prediction is unlikely because task difficulties well under 1 bit would be required. The general effect of the adjustments, as shown in Figure 7, is to increase low values of *ID*, thus further decreasing the likelihood of a negative prediction for *MT*.

Although it is interesting that *IP* decreases for each of the changes introduced, the magnitude of *IP* is less relevant to the present discussion than the overall accuracy of the model as determined by the statistical measures of correlation and standard error. The rate of *IP* = 8.2 bits/s for the first experiment is a full 2.4 bits/s lower than that found using Fitts' model; however, low rates often emerge, sometimes under 5 bits/s (e.g., Epps, 1986; Jagacinski, Repperger, Ward, & Moran, 1980; Kantowitz & Elvers, 1988; Kvålseth, 1977).

To conclude, the trend of increasing correlation and decreasing standard error progressing down the columns in Figure 6 within each experiment suggests that the adjustments introduced improve the model's accuracy.

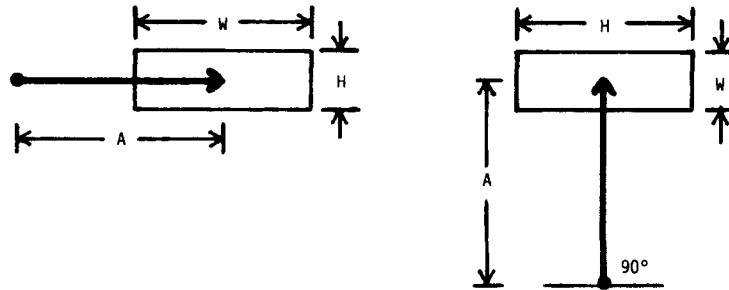
Figure 7. Scatter plot of MT versus ID . The data are from Fitts (1954); however, ID has been recalculated using W_e and a logarithmic expression based on Shannon's information theorem (see Equation 10).



3.6. Effective Target Amplitude

It follows from the preceding discussion that an adjustment may also be in order to reflect the actual distance moved, resulting in an "effective" target amplitude, A_e . The possibility seems strongest that $A_e < A$, particularly when A/W is small, but many factors are at work such as the type of input device and the control-display (CD) gain setting. The data in Fitts' report do not permit an investigation of this point; however, it was observed that, of the two possibilities, undershoot errors were more common (Fitts, 1954, p. 385). This

Figure 8. The changing roles of target width and target height as the approach angle changes.



trend has also been noted in other studies (Glencross & Barrett, 1983; P. A. Hancock & Newell, 1985, p. 159; Langolf et al., 1976; Wright & Meyer, 1983).

The implications of this are subtle and may be of little practical consequence. If a prediction equation is derived using adjusted amplitude measures (reflecting what subjects actually did) and then is applied in subsequent designs, there may be a systematic departure of performance from predictions. More errors may occur than predicted because output responses may not be a normally distributed reflection of input stimulus but may be skewed inward.

3.7. Targets and Angles

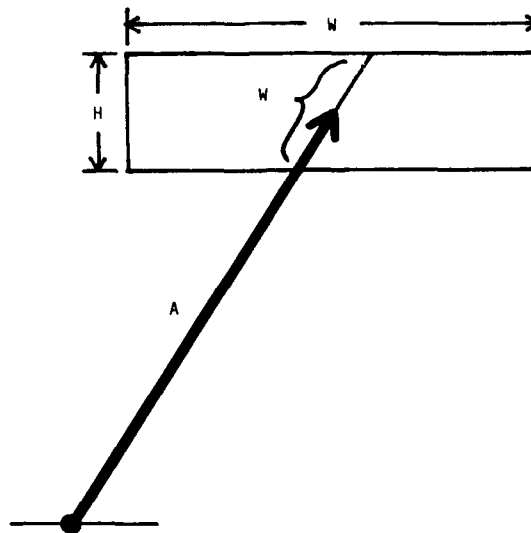
There are two aspects of dimensionality in Fitts' law tasks: the shape of targets and the direction of movement. When movements are limited to one dimension (e.g., back and forth) and both target height (H) and target width are varied, there is evidence that target height has only a slight main effect on movement time (Kvålseth, 1977; Salmoni, 1983; Welford, 1968, p. 149). Schmidt (1988, p. 278) noted that horizontal motion toward a target results in an elliptical pattern of hits, with the long axis on the line of approach.⁸

When the shape of the target and the direction of movement vary, the situation is confounded. For rectangular targets in two-dimensional (2D) positioning tasks, as the approach angle changes from 0° to 90° (relative to the horizontal axis), the roles of target width and target height reverse (see Figure 8).

Varying the direction of approach raises the question, What is target width? At the model-building stage, the issue is avoided somewhat by using W_e , as described earlier. Most likely, W_e should be derived from the endpoint variability in two dimensions, calculated in Cartesian coordinates from

⁸ The reader is invited to verify this with a felt-tipped pen and a sheet of paper. Tapping back and forth between two rectangular targets (as in Figure 1) will produce two patterns as described.

Figure 9. The relationship between target width and approach angle. A possible substitute for target width when the approach angle varies is the distance through the target along the approach vector (W').



$\sqrt{x^2 + y^2}$. Although this idea awaits empirical testing, it can extend to three-dimensional (3D) movements as well.

When a derived model is used for prediction in 2D movement tasks, the problem of target width must be addressed directly: What value for W should be used in calculating ID ? There are several possibilities. Considering first only rectangular targets, it is probably wrong to consistently use the horizontal extent of a target for W , because a wide, short target approached from above or below at close range will yield a negative ID (if Fitts' or Welford's formulation is used). This situation is common in text-selection tasks where wide, short targets (viz., words) are the norm. The text-selection experiments by Card et al. (1978) and Gillan et al. (1990) both cited experimental conditions yielding negative ID s.

Research on potential substitutes for target width is scarce. Possibilities include H , $W + H$, or $W \times H$ (Gillan et al., 1990). Perhaps the smaller of W or H is appropriate because the lesser of the two extents seems more indicative of the precision demands of the task. Another possibility is the span of the target along an approach vector through the center. This distance, W' , is shown in Figure 9. Although untested, the latter idea is appealing in that circles or other shapes of targets can be accommodated. It also has the advantage of maintaining the one dimensionality of the model.⁹

⁹ A test of two-dimensional models for Fitts' law can be found in MacKenzie and Buxton (in press).

Projecting 3D objects onto a 2D CRT display is common on today's bit-mapped graphic systems. It follows that input strategies are needed to facilitate 3D interaction. A first-order solution is to map a 2D device into the third dimension (e.g., Chen, Mountford, & Sellen, 1988; Evans, Tanner, & Wein, 1981). Recent techniques include direct manipulation with an input glove (Foley, 1987; Tello, 1988; Zimmerman, Lanier, Blanchard, Bryson, & Harvill, 1987) or maneuvering a mouse in three dimensions (Ware & Baxter, 1989). Although no studies to date have employed Fitts' law in 3D computer interaction tasks, a need may arise as this mode of interaction matures.

4. COMPETING MODELS

The ultimate reason for building models (e.g., human performance models) is that they facilitate the way we think about a problem. Models are neither right nor wrong; only through their utility do they muster support in the scientific community. Although unquestionably robust, the information-processing analogy in Fitts' law does not sit well for all.

Several competing and overlapping models, including Fitts' law, are at the forefront of current research pushing toward a general theory of motor behavior. The following paragraphs extend the belief that a general model of human movement should accommodate the extremes of temporal and spatial constraints in movement tasks. There are classes of movements (e.g., drawing) that at present lack a paradigm for performance modeling. A new model, perhaps incorporating Fitts' law, could fulfill this need.

4.1. The Linear Speed-Accuracy Tradeoff

Of considerable interest recently is the linear speed-accuracy tradeoff discovered by Schmidt and colleagues (Schmidt, Zelaznik, & Frank, 1978; Schmidt, Zelaznik, Hawkins, Frank, & Quinn, 1979). The tradeoff, formally the *impulse variability model*, forecasts that the standard deviation in endpoint coordinates (viz., accuracy) is a linear function of velocity, calculated as distance over time:

$$W_e = a + b A/MT. \quad (13)$$

It is interesting that Equation 13 and Fitts' law contain the same three parameters (with the difference that W_e is the standard deviation of endpoint coordinates in Equation 13 and is $4.133 \times SD$ in Fitts' adjusted model). Although Equation 13 can be rearranged with MT as the predicted variable, it is still fundamentally different from Fitts' law because the relationship is linear rather than logarithmic and because the information analogy is absent.

Another difference is the nature of the tasks suited to each. The linear speed-accuracy tradeoff is demonstrably superior to Fitts' law for "temporally constrained" tasks. The distinction is summarized as follows. Under spatial constraints, a move proceeds as quickly as possible and terminates within a defined region of space (target width). This applies to Fitts' tapping task. Under temporal constraints, a move proceeds as accurately as possible and terminates at a specified time. Targets are points or lines in temporally constrained tasks. Subjects strike the target on time and avoid being too fast or too slow.

In relation to computer input, temporally constrained tasks are of a different genre. They include, for example, capturing moving targets and real-time interaction (perhaps in a music performance system). The distinction between temporal and spatial constraints is by no means dichotomous. Drawing, tracing, and inking have features of both: A user moves a tracking symbol (cursor, cross, etc.) at an optimal velocity while attending to the accuracy demands of the task. How should such tasks be modeled? Is the focus on minimizing time (the dependent variable in Fitts' law) or on minimizing error (the dependent variable in Equation 13)?

The task of drawing is a simple example. The Keystroke-Level Model (Card et al., 1980) provides a rough estimate of the time to draw a series of line segments (t_D in ms) from the total length of the segments (l_D in cm) and the number of segments (n_D). The equation:

$$t_D = 900 n_D + 160 l_D \quad (14)$$

was offered as a restricted operator—dependent on the system, user, and device—and was included only to extend the generality of the Keystroke-Level Model to this class of movement tasks. Notably, accuracy is not represented in the equation. One may anticipate that a class of models for tasks with temporal constraints, such as drawing, may embody the linear speed-accuracy tradeoff given in Equation 13.

4.2. Power Functions

Several power functions have been proposed including the following general form (Kvålseth, 1980):

$$MT = a A^b W^c \quad (15)$$

A reanalysis of Fitts' (1954) data reveals that Equation 15 provides a higher multiple correlation (R) than the single-factor correlation (r) using Fitts' relationship. A test of positioning times using six cursor control devices also

showed higher correlations using Equation 15 (Epps, 1986). Note, however, that the improved fit is largely due to the extra degree of freedom. Equation 15 has three empirically determined constants; Fitts' law has two. And as noted earlier, a strength of Fitts' model is the physical interpretation afforded by the terms in the equation. A similar casting is difficult for a , b , and c in Equation 15.

Several permutations of Equation 15 are possible. If $b = -c$, then:

$$MT = a(A/W)^b. \quad (16)$$

Taking the base-2 logarithm of each side yields:

$$\begin{aligned} \log_2 MT &= \log_2 a + b \log_2(A/W) \\ &= a' + b \log_2(A/W), \end{aligned} \quad (17)$$

which is similar to Fitts' law except the log of movement time is the predicted variable (T. B. Sheridan & Ferrell, 1963).

Another permutation, introduced by Meyer et al. (1988), sets the exponent in Equation 16 to $1/2$ and positions slope and intercept coefficients in the usual place for linear regression:

$$MT = a + b\sqrt{A/W}. \quad (18)$$

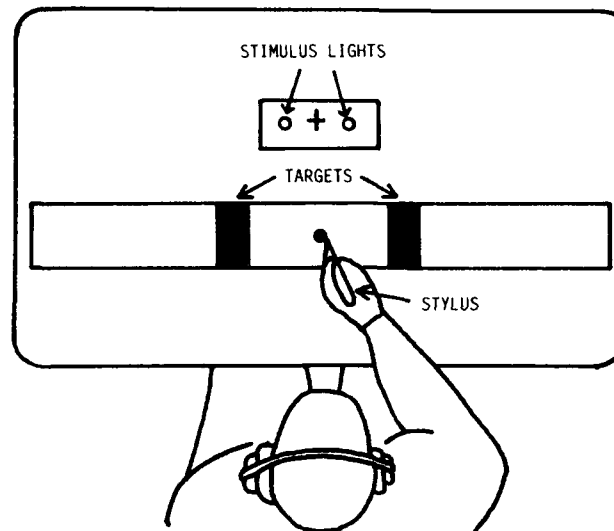
Equation 18, formally the *stochastic optimized-submovement model*, is supported by a comprehensive theory on the random variability of neuromotor force pulses. In a reanalysis of Fitts' (1954) data, higher correlations were found using Equation 18 than using Fitts' law (Meyer et al., 1988); however, they are not as high as those in Figure 6 using the Shannon formulation.

The model provides a unified conceptual framework encompassing both the linear speed-accuracy model and Fitts' log model. Meyer and colleagues found that movements following the Fitts' paradigm are composed of submovements with durations, distances, and endpoint distributions conforming to the linear speed-accuracy model. This is an important link. A goal of the stochastic optimized-submovement model is to reconcile the range of spatial and temporal demands in human movement in a general theory of motor behavior (Meyer et al., 1990). The promise for performance modeling of user input to computers is a single model capable of expressing a wider range of movement tasks.

5. APPLICATIONS OF FITTS' LAW

A comprehensive review of research applying Fitts' law in studies on human movement would be a monumental task. A quick tally from the *Social Sciences*

Figure 10. Discrete paradigm for Fitts' law experiments (after Fitts & Peterson, 1964).



Citation Index (SSCI) between 1970 and 1988 reveals 248 citations of Fitts' 1954 article. Even this is not fully indicative of the widespread use of Fitts' model because a large body of research in fields such as medicine, sports, and human factors is published in journals, books, and conference proceedings not surveyed in the *SSCI*. The following review is cursory and quickly proceeds to the relevant research in human factors and HCI.

5.1. The Generality of Fitts' Law

Building on Fitts' evidence that the rate of human information processing is constant across a range of task difficulties, other researchers adopted the model to determine *IP* in settings far removed from Fitts' original theme. It is evident in reviewing the literature that the new factors often confound the problem of measurement. Numerous studies report vastly different measures for very similar processes.

In a study similar to Fitts' initial report, Fitts and Peterson (1964) measured *IP* for a "discrete" task in which subjects responded to a stimulus light and tapped a target on the left or right. This experimental arrangement has been widely adopted in subsequent research (see Figure 10).

In comparison to $IP = 10.6$ bits/s for "serial" or reciprocal tapping tasks (Fitts, 1954), a rate of 13.5 bits/s was found for discrete tasks (after factoring out reaction time; Fitts & Peterson, 1964). It is interesting that a difference of 2.9 bits/s surfaced for two tasks that are essentially the same, except for the

serial versus discrete nature of the movements. Others have also found a higher *IP* for discrete tasks over serial tasks (Megaw, 1975; Sugden, 1980). Keele (1968) suggested that discrete tasks may yield a higher *IP* because they exclude time on target, unlike serial tasks.

Besides the large number of studies cited in the previous section that tested the validity of the model, many simply adopted the model as a tool to investigate other issues. The role of visual feedback in controlling the accuracy of movement has been the topic of many experiments using Fitts' law (e.g., Carlton, 1981; Crossman, 1960; Glencross & Barrett, 1983; Keele & Posner, 1968; Kvålseth, 1977; Meyer et al., 1988; Wallace & Newell, 1983). The usual method is to cut off visual feedback, a period of time after a movement begins, and compare the period of feedback deprivation with changes in accuracy, *MT*, or *IP*. It has been found that movements under approximately 200 ms are ballistic and not controlled by visual feedback mechanisms whereas those over 200 ms are.

Fitts' law has performed well for a variety of limb and muscle groups. High correlations appear in studies of wrist flexion and rotation (Crossman & Goodeve, 1963/1983; Meyer et al., 1988; Wright & Meyer, 1983), finger manipulation (Langolf et al., 1976), foot tapping (Drury, 1975), arm extension (B. A. Kerr & Langolf, 1977), head movement (Andres & Hartung, 1989; Jagacinski & Monk, 1985; Radwin, Vanderheiden, & Lin, 1990), and microscopic movements (W. M. Hancock, Langolf, & Clark, 1973; Langolf & W. M. Hancock, 1975). Underwater experiments have provided a platform for further verification of the model (R. Kerr, 1973; R. Kerr, 1978), as have experiments with mentally retarded patients (Wade et al., 1978), patients with Parkinson's disease (Flowers, 1976) or with cerebral palsy (Bravo, LeGare, Cook, & Hussey, 1990), the young (Jones, 1989; B. Kerr, 1975; Salmoni, 1983; Salmoni & McIlwain, 1979; Sugden, 1980; Wallace et al., 1978), and the aged (Welford et al., 1969). An across-species study verified the model in the movements of monkeys (Brooks, 1979). It has been suggested that the model would hold for the mouth or any other organ for which the necessary degrees of freedom exist and for which a suitable motor task could be devised (Glencross & Barrett, 1989).

Tabulating the results from these reports reveals a tremendous range of performance indices, from less than 1 bit/s (Hartzell, Dunbar, Beveridge, & Cortilla, 1983; Kvålseth, 1977) to more than 60 bits/s (Kvålseth, 1981). Most studies report *IP* in the range of 3 to 12 bits/s.

5.2. Review of Six Studies

Despite the large body of research evaluating the performance of computer input devices for a variety of user tasks, the discipline of HCI has not, as a

rule, been a proving ground for Fitts' law performance models. Most related HCI research uses "task completion time" as the unit of study, with errors or other measures reported in separate analyses. Two-factor repeated measures experiments with several levels each for task and device are the norm (e.g., Buxton & Myers, 1986; English, Engelbart, & Berman, 1967; Ewing, Mehrabanzad, Sheck, Ostroff, & Shneiderman, 1986; Goodwin, 1975; Gould, Lewis, & Barnes, 1985; Haller, Mutschler, & Voss, 1984; Karat, McDonald, & Anderson, 1984; Mehr & Mehr, 1972; Sperling & Tullis, 1988). See Greenstein and Arnaut (1988), Milner (1988), or Thomas and Milan (1987) for reviews of this body of research.

Six Fitts' law studies have been selected as relevant to the present discussion. These are surveyed in reference-list order, focusing initially on the methodology and empirical results. An assessment of the findings within and across studies is deferred to the end.

Card, English, and Burr (1978)

This highly cited work stands apart from other investigations by nature of its goal to transcend the simplistic ranking of devices and to develop models useful for subsequent device evaluations. The idea is that, once a model is derived, it can participate in subsequent designs by predicting performance in *different scenarios before design is begun*.

Selection time, error rates, and learning time were measured in a routine text-selection task using four devices: a mouse, an isometric joystick, step keys, and text keys. The step keys moved the cursor up, down, left, or right in the usual way, whereas the text keys advanced the cursor on character, word, or paragraph boundaries. The joystick controlled the velocity and direction of the cursor from the magnitude and direction of the applied force, with negligible displacement of the stick.

For each trial, subjects pressed the space bar, homed their hand on the cursor-control device, advanced the cursor to a word highlighted in a block of text, then selected the word by pressing a button or key. Experimental factors were device (four levels), distance to target (A s = 1, 2, 4, 8, and 16 cm), target size (W s = 1, 2, 4, and 10 characters; one character = 0.246 cm), approach angle (0° - 22.5° , 22.5° - 67.5° , and 67.5° - 90°), and trial block. ID s ranged from -0.14 bits (A = 1 cm, W = 10 characters) to 6.0 bits (A = 10 cm, W = 1 character)—The negative index is discussed later. Target height was held constant at 0.456 cm, the height of each character.

Using Welford's variation of Fitts' law, prediction equations were derived for the two continuous devices. The least-squares regression equation predicting MT (in ms) for the mouse was:

$$MT = 1030 + 96 ID, \quad (19)$$

with $IP = 10.4$ bits/s ($r = .91$, $SE = 70$ ms), and for the joystick,

$$MT = 990 + 220 ID, \quad (20)$$

with $IP = 4.5$ bits/s ($r = .94$, $SE = 130$ ms).

Mean MT was lowest for the mouse (1660 ms, $SD = 480$ ms) despite the fact that mean homing time was highest (360 ms, $SD = 130$ ms). The joystick was a close second ($MT = 1830$ ms, $SD = 570$ ms), followed by the text keys ($MT = 2260$ ms, $SD = 1700$ ms) and step keys ($MT = 2510$ ms, $SD = 1640$ ms).

Error rates ranged from 5% for the mouse to 13% for the step keys. Approach angle did not affect mean movement time for the mouse, but it increased movement time by 3% for the joystick when approaching a target along the diagonal axis.

Drury (1975)

Welford's variation of Fitts' law was evaluated as a performance model in a study of foot pedal design. Using their preferred foot, subjects tapped back and forth between two pedals for 15 cycles (30 taps). Six different amplitudes ($As = 150, 225, 300, 375, 525$, and 675 mm) were crossed with two pedal sizes ($Ws = 25$ and 50 mm). The mean width of subjects' shoes (108.8 mm) was added to target width as a reasonable adjustment because any portion of a shoe touching the target was recorded as a hit. As such, ID s ranged from 0.53 to 2.47 bits. With $A = 150$ mm and $W = 50 + 108.8 = 158.8$ mm, the task difficulty was calculated as $\log_2(150.0/158.8 + 0.5) = 0.53$ bits. This is an extremely relevant example of a task condition in which an index of difficulty less than 1 bit is perfectly reasonable. In effect, the targets were overlapping.

The correlation between MT and ID was high ($r = .970$, $p < .01$), with regression line coefficients of 187 ms for the intercept and 85 ms/bit for the slope ($IP = 11.8$ bits/s). Overall error rates were not reported, but blocks with more than one miss were repeated; thus, by design, the error rate was less than 3.3%.

Epps (1986)

Six cursor-control devices were compared in a target-selection task with performance models derived using Fitts' law, a power model (Equation 15), and the following first-order model proposed by Jagacinski, Repperger, Ward, and Moran (1980):

$$MT = a + bA + c(1/W - 1). \quad (21)$$

Device types included two touchpads (relative and displacement), a trackball, two joysticks (displacement and force; both with velocity control), and a mouse. For each trial, subjects moved a cross-hair tracker to a randomly positioned rectangular target and selected the target by pressing a button. Target distance varied across four levels ($As = 2, 4, 8, \text{ and } 16 \text{ cm}$) and target size across five levels ($Ws = 0.13, 0.27, 0.54, 1.07, \text{ and } 2.14 \text{ cm}$), yielding ID s from 0.90 to 6.94 bits.

The power model provided the highest (multiple) correlation with MT across all devices, with the first-order model providing higher correlations for some devices but not others. The correlations throughout were low, however, in comparison to those usually found. Using Fitts' equation, r ranged from .70 for the relative touchpad to .93 for the trackball. Intercepts varied from -587 ms (force joystick) to 282 ms (trackball). The values for IP , ranging from 1.1 bits/s (displacement joystick) to 2.9 bits/s (trackball), are among the lowest to appear in Fitts' law experiments.

If an error was committed, subjects repositioned the cursor inside the target and pressed the select button again. Although the frequency of this behavior was not noted, presumably these trials were entered in the analysis using the total time for the operation.

Jagacinski and Monk (1985)

Fitts' law was applied to a target-acquisition task using a displacement joystick for position control and a head-mounted sight using two rotating infrared beams. Each trial began with the cursor in the middle of the display and the appearance of a circular target on the screen. Subjects moved the cursor to the target and selected it. On-target dwell time (344 ms), rather than a button push, was the criterion for target selection.

Experimental factors were device (two levels), target distance ($As = 2.45^\circ, 4.28^\circ, \text{ and } 7.50^\circ$ of visual angle), target size ($Ws = 0.30^\circ, 0.52^\circ, \text{ and } 0.92^\circ$ for the joystick; $Ws = 0.40^\circ, 0.70^\circ, \text{ and } 1.22^\circ$ for the helmet-mounted sight), and approach angle ($0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, \text{ and } 315^\circ$). Task difficulties ranged from 2.0 to 5.6 bits for the helmet-mounted sight. Correlations between MT and ID were very high ($r = .99$) for both devices, with regression coefficients for the intercept of -268 ms (helmet-mounted sight) and -303 ms (joystick). The regression line slope for both devices was 199 ms/bit ($IP = 5 \text{ bits/s}$). Mean MT s were slightly longer along the diagonal axes for the joystick (7.2%) and for the helmet-mounted sight (9.1%). Because the selection criterion was dwell time inside the target, errors could not occur.

Kantowitz and Elvers (1988)

Fitts' law was evaluated as a performance model for two isometric joysticks—one for cursor position control, the other for cursor velocity

control. Each trial began with the appearance of a square target in the center of the screen and an asterisk pre-cursor on either side that tracked the applied force of the joystick. When the pre-cursor changed to a cross-hair cursor, the subject moved it to the target and selected the target. A trial terminated if the cursor remained stationary (± 3 pixels) for 333 ms, the horizontal direction of movement changed, or 4 s elapsed. Experimental factors were device (two levels), target distance (A s = 170, 226, and 339 pixels), target size (W s = 20 and 30 pixels), and CD gain (high and low). Four target distance/size combinations were chosen with ID s ranging from 3.5 to 5.5 bits.

The velocity-control joystick regression line had a steeper slope, and therefore a lower IP , than the position-control joystick (IP s = 2.2 bits/s vs. 3.4 bits/s, respectively). There was no main effect for CD gain; for each device, the high and low gain regression lines were parallel. The intercepts, however, were large and negative. Under high-gain and low-gain conditions, respectively, intercepts were -328 and -447 ms under position control and -846 and -880 ms under velocity control. Correlations ranged from .62 to .85. The average error rate was very high (around 25%), although figures were not provided across factors.

Ware and Mikaelian (1987)

Welford's variation of Fitts' law was applied to positioning data from a selection task using an eye tracker (Gulf and Western series 1900). A cross-hair cursor positioned on a CRT display was controlled by the reflection from subjects' cornea of an infrared source. Targets were selected by three methods: a hardware button, dwell time on target (400 ms), or an on-screen button. Seven rectangles (3.2 cm \times 2.6 cm) were presented to the subjects in a vertical row. After fixating on the center rectangle for 0.5 s, one of the seven became highlighted, whereupon subjects immediately fixated on it and selected it.

The application of Fitts' law in this study is weak. Target size was kept constant (2.6 cm), but distance was varied over four levels (0, 2.6, 5.2, and 7.8 cm). Although ID s ranged from -1.0 bit to 1.8 bits, no rationale was provided for the negative index at $A = 0$ cm, calculated as $\log_2(0/2.6 + 0.5) = -1$ bit.¹⁰ Correlations and regression coefficients were omitted in lieu of a scatter plot of MT versus ID with regression lines for each selection technique. For the purpose of this survey, equations were inferred from the plots. Intercepts ranged from 680 to 790 ms, and slopes ranged from 73 to 107 ms/bit. The highest IP was for the hardware button condition (13.7 bits/s), and the lowest was for dwell time (9.3 bits/s).

Error rates were high, ranging from 8.5% (hardware button) to 22% (on-screen button). As the investigators noted, an eye tracker can provide fast

¹⁰ Note that the unusual choice of $A = 0$ as an experimental condition precludes the use of Fitts' equation, because $\log_2 0$ is undefined.

cursor positioning and target selection, as long as accuracy demands are minimal.

5.3. Across-Study Comparison of Performance Measures

We now proceed with the task of assessing the findings and comparing them across studies. Figure 11 tabulates for each device condition the regression coefficients, the *MT-ID* correlation, and the percentage errors. Both the slope and *IP* are provided for convenience, as are the values from Fitts' (1954) tapping experiment with a 1-oz stylus (see Figures 2 and 3). The entries are ordered by decreasing *IP*. This is not the same as ordering by increasing *MT* because the intercepts also contribute to *MT*. It is felt that *IP* is more indicative of the overall performance of a device and that normalizing the intercepts is reasonable for this comparison.

The presence of nine negative intercepts in Figure 11 is the first sign of trouble. A negative intercept implies that, as tasks get easier, a point is reached where the predicted movement time is negative. This, of course, is nonsense and indicates a flaw in the application of the model or the presence of uncontrolled variations in the data. Beyond this, the most notable observation is the overall lack of consensus in the measures. The spread of values is astonishing: Performance indices range from 1.1 to 13.7 bits/s, and intercepts range from -880 to 1030 ms. These values, however, probably do not truly reflect the innate differences in the devices. Although differences are expected across devices, similar measures should emerge in the figure where different entries are for the same device.

For example, the mouse was evaluated by Card et al. (1978) and Epps (1986). The former cite *IP* = 10.4 bits/s whereas the latter cites *IP* = 2.6 bits/s. These values differ by a factor of four! Also, the intercepts differ by 922 ms. So, what is the Fitts' law prediction equation for the mouse? The answer is up for debate.

Also, an isometric, velocity-control joystick was tested by Card et al. (1978), Epps (1986), and Kantowitz and Elvers (1988). Again, the outcome is disturbing. In the order just cited, the intercepts were reported as 990, -587, and 863 ms (average), and *IP* was reported as 4.5, 1.2, and 2.2 bits/s. It seems that the goal cited earlier—to develop models for evaluating devices and interaction techniques prior to implementation—remains elusive.

5.4. Sources of Variation

We can attempt to reconcile the differences by searching out the major sources of variation. Indeed, some of these are nascent traits of direct manipulation systems (rather than quirks in methodology) and, therefore, are particularly pertinent to the context of HCI. Identifying these provides a basis for evaluating and comparing studies. When disparities emerge, it may be

Figure 11. Survey of Fitts' law performance characteristics from six studies on user input devices.

Device	Study	Regression Coefficient ^a			r	Errors (%)	Comments
		Intercept, a (ms)	Slope, b (ms/bit)	IP (bits/s)			
Eye tracker ^b	Ware & Mikaelian (1987)	680	73	13.7	—	8.5	Hardware button
Foot pedal	Drury (1975)	187	85	11.8	.97	< 3.3	Experiment 2
Hand ^c	Fitts (1954)	12.8	94.7	10.6	.98	1.8	Tapping, 1-oz stylus
Mouse	Card, English, & Burr (1978)	1030	96	10.4	.91	5	
Eye tracker ^b	Ware & Mikaelian (1987)	790	97	10.3	—	22	On-screen button
Eye tracker ^b	Ware & Mikaelian (1987)	680	107	9.3	—	12	Dwell time
Helmet sight	Jagacinski & Monk (1985)	—268	199	5.0	.99	0	
Joystick	Jagacinski & Monk (1985)	—303	199	5.0	.99	0	Isometric; position control
Joystick	Card, English, & Burr (1978)	990	220	4.5	.94	12	Isometric; velocity control
Joystick	Kantowitz & Elvers (1988)	—328	297	3.4	.62	25	Isometric, position, high gain
Joystick	Kantowitz & Elvers (1988)	—447	297	3.4	.76	25	Isometric, position, low gain
Trackball	Epps (1986)	282	347	2.9	.93	0	
Mouse	Epps (1986)	108	392	2.6	.83	0	
Touchpad	Epps (1986)	181	434	2.3	.74	0	Absolute positioning
Joystick	Kantowitz & Elvers (1988)	—846	449	2.2	.84	25	Isometric, velocity, high gain
Joystick	Kantowitz & Elvers (1988)	—880	449	2.2	.85	25	Isometric, velocity, low gain
Touchpad	Epps (1986)	—194	609	1.6	.70	0	Relative positioning
Joystick	Epps (1986)	—587	861	1.2	.81	0	Isometric; velocity control
Joystick	Epps (1986)	—560	919	1.1	.86	0	Displacement; velocity control

^a $MT = a + b ID$; $IP = 1/b$. ^bData inferred from plot. ^cProvided for comparison purposes only.

possible to adjust measures or to predict comparative outcomes under hypothetical circumstances.

Device Differences

If the research goal is to establish a Fitts' law (or other) performance model for two or more input devices, then the only source of variation that is desirable is the between-device differences. This is what the investigations are attempting to measure. Accomplishing this assumes, somewhat unrealistically, that all other sources of variation are removed or are controlled for. Of course, very few studies are solely interested in device differences. *Sources of variation* become *factors* in many studies—equally as important to the research as model fitting across devices.

We can cope with the disparity in Figure 11 by looking for across-study agreement on within-study ranking rather than comparing absolute measures. The mice and velocity-control isometric joysticks evaluated by Card et al. (1978) and Epps (1986) provide a simple example. The index of performance was higher for the mouse than for the joystick *within* each study. One could conclude, therefore, that the mouse is a better performer (using *IP* as the criterion) than the joystick, even though the absolute values are deceiving. (Note that the joystick in Card et al.'s study yielded a higher *IP* than the mouse in Epps' study.) Furthermore, the differences between devices expressed as a ratio was about the same: *IP* was higher for the mouse than for the joystick by a factor of $10.4/4.5 = 2.3$ in Card et al.'s (1978) study and by a factor of $2.6/1.2 = 2.2$ in Epps' (1986) study.

Just as the units disappear when the ratio of the performance indices is formed, so too may systematic effects from other sources of variation, including a myriad of unknown or uncontrolled factors present in an experiment. Indeed, experiment differences are evident in Figure 11: Epps' (1986) and Kantowitz and Elvers' (1988) studies showed low values for *IP*, whereas Card et al.'s (1978) and Drury's (1975) studies showed high values. Thus, relative differences within studies gain strength if across-study consensus can be found.

A larger sample of studies would no doubt reveal across-study consensus on other performance differences. The performance increment found in Kantowitz and Elvers' (1988) study for the position-control joystick over the velocity-control joystick, to cite one example, was noted in another study not in the survey (Jagacinski, Repperger, Moran, Ward, & Glass, 1980).¹¹

We should acknowledge as performance determinants the range of muscle

¹¹ The ratio of performance differences was also the same: *IP* for the position-control system was higher than *IP* for the velocity-control system by a factor of $3.2/2.2 = 1.5$ in Kantowitz and Elvers' (1988) study and by a factor of $5.9/3.9 = 1.5$ in Jagacinski, Repperger, Moran, Ward, and Glass' (1980) study. (In the latter study, the values cited were averaged over the dwell time and steadiness criteria for target capture.)

and limb groups engaged by different manipulators. Because smaller limb groups (e.g., wrist vs. arm) have shown higher ratings for *IP* (Langolf et al., 1976), performance increments are reasonable when complex arm movements are avoided. With fewer degrees of freedom for the head or eyes than for the arm, the relatively high rates for the eye tracker and helmet-mounted sight in Figure 11 may be warranted. This does not, however, account for the high ranking of the foot pedals.

It is felt that Fitts' law performance differences can be attributed to other characteristics of devices, such as number of spatial dimensions sensed (one, two, or three) or property sensed (pressure, motion, or position); however, our sample is too small to form a basis for generalization. Besides, the studies surveyed may contain stronger sources of variation.

Task Differences

It is naive, perhaps, to suggest that there exists a generic task that can accommodate simple adjustments for other factors, such as device. One might argue that Fitts' tapping task is remote and inappropriate: It is not a particularly common example of user interaction with computers. Its one-dimensional simplicity, however, has advantages for model building, not the least of which is access to a substantial body of research. For example, there is evidence that a serial task yields an index of performance 2 to 3 bits/s lower than a similar discrete task (e.g., Fitts & Peterson, 1964). Discrete tasks may be more akin to direct manipulation systems, but experiments are easier to design and conduct using a serial task. Knowledge of a 2- to 3-bit/s increment for discrete operation after conducting a serial task experiment is a valuable resource for researchers.

Of the six studies surveyed, all but one used a discrete task. Drury's (1975) serial foot-tapping experiment yielded $IP = 11.8$ bits/s, but it may have shown a rate around 14 bits/s had a discrete task been used. Although this would tend to disperse further the rates in Figure 11, indices in the 15- to 20-bits/s range are not uncommon in Fitts' law studies.

Five of the six studies used a simple target-capture task, and one (Card et al., 1978) used a text-selection task. The cognitive load on subjects may have been higher in the latter case due to the presence of additional text on the screen. Perhaps the burden of finding and keeping track of highlighted text within a full screen of text continued throughout the move. This task difference would reduce performance, but one can only speculate on where the effect would appear. The evidence leans toward the intercepts because they were highest in this study (1030 and 990 ms).

Selection Technique

The method of terminating tasks deserves separate analysis from other aspects of tasks. In the studies by Card et al. (1978) and Epps (1986), the

target-selection button for all devices except the mouse was operated with the opposite hand from that which controlled the device. Ware and Mikaelian (1987) also used a separate hand-operated button as one of the selection conditions with the eye tracker. There is evidence that task completion times are reduced when a task is split over two hands (e.g., Buxton & Myers, 1986), suggesting that parallel cognitive strategies may emerge when positioning and selecting are delegated to separate limbs. This may explain the trackball's higher *IP* over the mouse in Epps' (1986) experiment – The mouse task was one-handed, the trackball task was two-handed. Unfortunately, this speculation does not extend to Card et al.'s (1978) study where *IP* was significantly higher for the mouse (one-handed) than for the joystick (two-handed).

Conversely, and as mentioned earlier, target-selection time may be additive in the model, contributing to the intercept of the regression line but not to the slope. This argument has some support in Epps' (1986) study where the intercept is second highest out of five for the mouse, where an additive effect would appear. Both the mouse and the joystick yielded similar intercepts in Card et al.'s (1978) study, thus lending no support either way.

There are presently versions of each device that permit device manipulation and target selection with the same limb. Therefore, a Devices \times Mode of Selection experiment could examine these effects on the intercept and slope in the prediction equation. In fact, mode of selection was a factor in Ware and Mikaelian's (1987) study. Based on this study, one would conclude that *IP* increases when selection is delegated to a separated limb (as it did for the hardware button condition vs. the dwell time or on-screen button conditions; see Figure 11).

Range of Conditions and Choice of Model

In Fitts' (1954) tapping experiments, subjects were tested over four levels each for target amplitude and target width with the lowest value for target amplitude equal to the highest value for target width (see Figure 2). In all, subjects were exposed to 16 *A-W* conditions with *ID*s ranging from 1 to 7 bits. Figure 12 tabulates the range of target conditions employed in the studies surveyed.

Some stark comparisons are found in Figure 12. Kantowitz and Elvers (1988) and Ware and Mikaelian (1987) limited testing to four *A-W* conditions over a very narrow range of *ID*s (2.00 bits and 2.80 bits, respectively). Although Drury (1975) used 12 *A-W* conditions, the range of *ID*s was only 1.94 bits. This resulted because the spreads for *A* and *W* were small. Despite using six levels for *A*, the ratio of the highest value to the lowest value was only 4.5, and the same ratio for *W* was only 1.2. (When a scatter plot is limited to a very narrow range, one can imagine a line through the points tilting to and fro with a somewhat unstable slope!) The narrow range of *ID*s in Kantowitz and Elvers' (1988) study, combined with the observation that the lowest *ID*

Figure 12. Summary of target conditions used in six Fitts' law studies.

Study	Amplitudes (High/Low)	Widths (High/Low)	Number of A-W Conditions	Index of Difficulty (Bits)		
				Low	High	Range
Card, English, & Burr (1978) ^a	1, 2, 4, 8, 16 cm (16)	1, 2, 4, 10 characters ^b (10)	20	-0.14	6.03	6.18
Drury (1975) ^a	150, 225, 300, 375, 525, 675 mm (4.5)	133.8, 158.8 mm (1.2)	12	0.53	2.47	1.94
Epps (1986)	2, 4, 8, 16 cm (8)	0.13, 0.27, 0.54, 1.07, 2.14 cm (16)	9	0.90	6.94	6.04
Jagacinski & Monk (1985)	2.45°, 4.28°, 7.50° visual angle (3.1)	0.30°, 0.52°, 0.92° visual angle for joystick (3.1) 0.40°, 0.70°, 1.22° visual angle for helmet (3.1)	9	2.41	5.64	3.22
Kantowitz & Elvers (1988)	170, 226, 339, 453 pixels (2.7)	20, 30 pixels (1.5)	4	3.50	5.50	2.00
Ware & Mikaelian (1987) ^a	0, 2.6, 5.2, 7.8 cm (-)	2.6 cm (1)	4	-1.00	1.80	2.80
Fitts (1954) ^c	2, 4, 8, 16 in. (8)	¼, ½, 1, 2 in. (8)	16	1.00	7.00	6.00

^aWelford's formulation used for calculation of *ID* (see Equation 8). ^bOne character = 0.246 cm. ^cIncluded for comparison purposes only.

was very high (3.50 bits), could explain the large negative intercepts. (After traveling 3.5 bits to the origin, a line swiveling about a narrow cluster of points could severely miss its mark!) It is also worth noting that the predicted movement time when $ID = 1$ bit with Kantowitz and Elvers' (1988) velocity-control joystick under low-gain conditions is $449(1) - 880 = -431$ ms. Although $ID = 1$ bit is not unreasonable (e.g., see Figure 2), a negative prediction for movement time is. Had this experiment included a wider range of ID s, extending down to around 1 bit, the regression line intercepts would have been higher and the slopes would have been lower.

Card et al. (1978) and Epps (1986) used a reasonable number of conditions (20 and 9, respectively) over a wide range of task difficulties (6.18 and 6.04 bits, respectively). These represent a strong complement of conditions that should yield results bearing close scrutiny.

Although a nonzero intercept can be rationalized a variety of ways, the studies by Card et al. (1978) and Ware and Mikaelian (1987) present a special problem. In these, ID s < 0 bits represent conditions that *actually occurred*; thus, it is certain that an appreciable positive intercept results. A contributing factor in the Card et al. (1978) study is the confounding approach angle (discussed later). In both studies, however, the negative ID s would disappear simply by using Shannon's formulation for ID (Equation 10). This would reduce the regression line intercepts because the origin would occur left of the tested range of ID s (where it should) rather than in the middle.

It is also possible that Fitts' law is simply the wrong model in some instances. Card et al. (1978) noted in the scatter plot for the joystick a series of parallel lines for each target amplitude condition. Certainly, this is not predicted in the model: A and W play equal but inverse roles, and, at a given ID , only random effects should differentiate the outcomes. Noting the systematic effect of amplitude, separate prediction equations were devised for each value of A . The result was a series of parallel regression lines with slopes around 100 ms/bit ($IP = 10$ bit/s) and with intercepts falling as A decreased. With this adjustment, the joystick and mouse IP s were about the same. However, this is a peculiar situation for the model—In essence, target amplitude ceases to participate.

The range of conditions also bears heavily on the coefficient of correlation. Although r is extremely useful for comparisons within a study, across-study comparisons are all but impossible unless the conditions are the same. It can be shown statistically that correlations are uncharacteristically low when a sample is drawn from a restricted range in a population (Glass & Hopkins, 1984, p. 92). This could explain the relatively low correlations in Figure 11 for Kantowitz and Elvers' (1988) study.

The extent of data aggregation also affects r . In the vast majority of Fitts' law studies, movement times are averaged across subjects and a single data point is entered into the analysis for each A - W condition. Epps (1986) did not

average across subjects and entered 240 data points into the analysis (12 Subjects \times 5 Amplitudes \times 4 Widths). The extra variation introduced is likely the cause of the relatively low correlations in this study.

Approach Angle and Target Width

In the Card et al. (1978) study, the use of approach angle as an experimental factor in conjunction with the consistent use of wide, short targets (viz., words) provides the opportunity to address directly a theoretical point raised earlier: What is target width when the approach angle varies?

When target distance was 1 cm and target width was 10 characters (2.46 cm), *ID* was calculated in this study using Welford's formulation as $\log_2(A/W + 0.5) = \log_2(1/2.46 + 0.5) = -0.14$ bits. This troublesome value, although not explicitly cited, appeared in the scatter plot of *MT* versus *ID* (Figure 6, p. 609). Because character height was 0.456 cm, a better measure of *ID* may have been $\log_2(1/0.456 + 0.5) = 1.43$ bits.¹² With a slope of 96 ms/bit for the mouse regression line, this disparity in *ID*s increases the intercept by as much as $[1.43 - (-0.14)] \times 96 = 151$ ms. The contribution could be even more in a regression analysis using W_e because adjusting target width generally increases the regression line slope and increases *ID* for easy tasks (see Figures 6 and 7).

Thus, the negative *ID*s and the very large intercepts in the Card et al. (1978) study are at least partially attributable to the one-dimensional limitations in the model and to the use of a formulation for *ID* that allows for a negative index of task difficulty. As the investigators noted, however, the time spent in grasping the device at the beginning of a move and the time for the final button-push were also contributing factors.

Epps (1986) and Jagacinski and Monk (1985) also varied approach angle. Because the targets were squares or circles, however, there is no obvious implication to the calculation of task difficulty or to the regression coefficients.

Error Handling

Response variability (viz., errors) is an integral part of rapid, aimed movements. Unfortunately, the role of accuracy is often neglected in the application of Fitts' law. Jagacinski (1989, p. 139) noted the following:

It is difficult to reach any conclusions when one system has exhibited faster target acquisitions, but has done so with less accuracy in comparison with another system. Both systems might have the same

¹² Using the Shannon formulation (Equation 10), the index of task difficulty under these conditions is further increased: $ID = \log_2(1/0.456 + 1) = 1.67$ bits.

speed-accuracy function, and the experimental evaluation might have simply sampled different points along this common function.

A simple way out of this dilemma is to build the model using W_e in the calculation of ID . The adjustment normalizes target width for a nominal error rate of 4%, as described earlier. However, none of the studies surveyed included the adjustment. Unfortunately, post hoc adjustments cannot be pursued at this time because error rates across levels of A and W were not reported. Although speculation is avoided on possible adjustments to the regression coefficients, it is instructive to review the strategies adopted for error handling.

Card et al. (1978) excluded error trials from the data analysis. Drury (1975) included errors in the data analysis however, only 1 miss in a block of 30 trials was permitted; otherwise, the block was repeated. Although Kantowitz and Elvers (1988) and Ware and Mikaelian (1987) reported very high error rates (up to 25%), it was not stated if error trials were included in the regression analyses—presumably they were.

In Kantowitz and Elvers' (1988) study, subjects were not allowed to reverse the horizontal direction of movement. If a reversal was detected, the trial immediately terminated and a miss was recorded if the cursor was outside the target. This precludes potential accuracy adjustments at the end of a trial, which, no doubt, would increase movement time.

Jagacinski and Monk (1985) and Epps (1986) introduced selection criteria whereby errors could not occur—A trial continued until the target was captured. If the cursor was outside the target when the select button was pressed, subjects in Epps' (1986) study repositioned the cursor and reselected the target. Although the frequency was not reported, the inclusion of trials exhibiting such behavior is most unusual.

Learning Effects

Learning effects are a nagging, ubiquitous source of variation in experiments that seek to evaluate "expert" behavior. Often, research pragmatics prevent practicing subjects up to expert status. Fortunately, Fitts' serial or discrete paradigm is extremely simple, and good performance levels are readily achieved. Of the six studies surveyed, three made no attempt to accommodate learning effects. Of those that did, each used a different criterion to establish when asymptotic or expert levels were reached. The most accepted statistical tool for this is a multiple comparisons test (e.g., Newman-Keuls, Scheffé, or Tukey) on mean scores over multiple sessions of testing (Glass & Hopkins, 1984, chap. 17).

Only Epps (1986) included such a test. Although the design was fully within subjects, only 2 hr of testing were needed for each subject. This was sufficient to cross all levels of device (six), session (five), amplitude (four), and width

(five). Subjects were novices and were given only two repetitions of each experimental condition; yet a multiple comparisons test (Bonferroni's *t* test; see Glass & Hopkins, 1984, p. 381) showed no improvement overall or for any device after the second session. The data analysis was based on Sessions 3 to 5.

Jagacinski and Monk (1985), who practiced subjects for up to 29 days, used the criterion of 4 successive days with the block means on each day within 3.5% of the 4-day mean. Card et al. (1978) developed a similar criterion based on a *t* test.¹³

5.5 Summary

Other sources of variation abound. Among these are the instructions to subjects, the number of repetitions per condition, the order of administering devices, the sensitivity of device transducers, the resolution and sampling rate of the measuring system, the update rate of the output display, and CD gain. However, our analysis will not extend further. The discussions on error handling and learning effects highlighted the vastly different strategies employed by researchers, but speculating on the effect of these in the model is digressive. These and other sources are felt to introduce considerable variation but with effects that are, for the most part, random. Systematic effects may be slight, unanticipated, or peculiar to one design.

The range of conditions selected at the experimental design stage is a major source of variation in results. Experiments can benefit by adopting a wide and representative range of *A-W* conditions (e.g., 1 to 7 bits; see Figure 2). This done, the investigators can proceed to build valid information-processing models when other factors such as device or task are added. Adopting the Fitts paradigm for serial tasks (Figure 1) or discrete tasks (Figure 10) offers the benefit of a simple experimental setup and invites access to a large body of past research.

Experiments that vary approach angle and use rectangular or other long and narrow targets can avoid a negative task difficulty by using the Shannon formulation of Fitts' law (see Equation 10 and Figure 4) and/or by adopting a new notion of target width in the calculation of *ID* (see Figures 8 and 9). Extending the model to accommodate varying approach angles and target shapes is one area in need of further research, particularly in light of the 2D nature of user input tasks on computers.

The variety of schemes to terminate tasks and select or acquire a target undoubtedly affects the outcome of a regression analysis. There are reason-

¹³ *t* tests are not as reliable as multiple comparisons tests when more than two means (blocks) are compared. The alpha (probability of a Type I error) associated with the *t* statistic is higher than predicted (Glass & Hopkins, 1984, p. 369).

able grounds for expecting a distinct, additive component to appear in the intercept; however, evidence is scant and inconclusive. Further research is needed.

A major deficiency in the application of Fitts' law in general is the absence of a sound and consistent technique for dealing with errors. Although not demonstrated in any of the studies surveyed, the model can be strengthened using the effective target width in calculating ID (see Figure 5). Doing so normalizes response variability (viz., errors) for a nominal error rate of 4%. This theoretically sound (and arguably vital) adjustment delivers consistency and facilitates across-study comparisons. The adjustment can proceed using the error rate or the standard deviation in endpoint coordinates, as shown earlier.

Experiments are strengthened by practicing subjects until a reasonable criterion for expert performance is met. The three studies that tested for learning effects did so using mean movement times. It may be appropriate to also test subjects' rate of information processing (viz., IP) as a criterion variable. This test could be strengthened using W_e in the calculation of ID (see Figure 2) to accommodate both the speed and accuracy of performance. The direct method of calculating IP (viz., $IP = ID/MT$; see Figure 2) is easier and probably better because it nulls the intercept, blending the effects into IP . This would accommodate separate, distinct learning effects for the intercept that would be unaccounted for if $IP = 1/b$ (from a regression analysis) were used.

The prediction equations in the Fitts' law studies surveyed reveal large inconsistencies, making it difficult to summarize and offer de facto standard prediction equations for any of the devices tested. Despite high correlations (usually taken as evidence of a model's worth), the failings in across-study comparisons demonstrate that extracting a Fitts' law prediction equation out of a research article and embedding it in a design tool may be premature as yet.

6. CONCLUSIONS

As human-machine dialogues evolve and become more direct, the processes and limitations underlying our ability to execute rapid, precise movements emerge as performance determinants in interactive systems. Powerful models such as Fitts' law can provide vital insight into strategies for optimal performance in a diverse design space.

This article has examined the theory, prediction power, and relevance of Fitts' law, citing its limitations and suggesting improvements. A survey of six studies has shown that applying the model and obtaining consistent results is not easy. Major sources of variation have been identified, and some suggestions in experimental design and methodology have been offered to

ensure the worth of future studies. Certainly more research—much more—is due. It has been shown that a post hoc adjustment on target width (for a nominal error rate of 4%) and the use of a more theoretically sound equation (Equation 10) can improve the power of the model and avoid erroneous predictions. These issues plus a proper understanding of device-task associations play a vital role in the development of Fitts' law performance models capable of participating in the design of natural, efficient human-computer interfaces.

Acknowledgments. I am grateful to Bill Buxton, Gil Hamann, Marilyn Mantei, Bob McLean, and members of the Input Research Group at the University of Toronto for their support and assistance and to the *HCI* referees who provided a meticulous critique of an earlier draft of this article. In particular, my review editor Stuart Card offered many helpful and challenging suggestions that brought perspective to many of the issues.

REFERENCES

- Andres, R. O., & Hartung, K. J. (1989). Prediction of head movement time using Fitts' law. *Human Factors*, 31, 703-713.
- Bainbridge, L., & Sanders, M. (1972). The generality of Fitts' law. *Journal of Experimental Psychology*, 96, 130-133.
- Beggs, W. D. A., Graham, J. C., Monk, T. H., Shaw, M. R. W., & Howarth, C. I. (1972). Can Hick's law and Fitts' law be combined? *Acta Psychologica*, 36, 348-357.
- Beggs, W. D. A., & Howarth, C. I. (1970). Movement control in a repetitive motor task. *Nature*, 225, 752-753.
- Bravo, P. E., LeGare, M., Cook, A. M., & Hussey, S. M. (1990). Application of Fitts' law to arm movements aimed at targets in people with cerebral palsy. In J. J. Presperin (Ed.), *Proceedings of the 13th Annual Conference of the Rehabilitation Engineering Society of North America* (pp. 21.3-21.4). Washington, DC: RESNA.
- Brooks, V. B. (1979). Motor programs revisited. In R. E. Talbott & D. R. Humphrey (Eds.), *Posture and movement* (pp. 13-49). New York: Raven.
- Buck, L. (1986). Target location effects in tapping tasks. *Acta Psychologica*, 62, 1-13.
- Buxton, W., & Myers, B. A. (1986). A study in two-handed input. *Proceedings of the CHI '86 Conference on Human Factors in Computing Systems*, 321-326. New York: ACM.
- Card, S. K., English, W. K., & Burr, B. J. (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics*, 21, 601-613.
- Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23, 396-410.
- Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Carlton, L. G. (1981). Processing visual feedback information for movement control. *Journal of Experimental Psychology: Human Perception and Performance*, 7, 1019-1030.
- Chen, M., Mountford, S. J., & Sellen, A. (1988). A study in interactive 3-D rotation

- using 2-D control devices. *Computer Graphics*, 22(4), 121-129.
- Crossman, E. R. F. W. (1960). The information-capacity of the human motor-system in pursuit tracking. *Quarterly Journal of Experimental Psychology*, 12, 1-16.
- Crossman, E. R. F. W., & Goodeve, P. J. (1983). Feedback control of hand-movement and Fitts' law. Reprinted in *Quarterly Journal of Experimental Psychology*, 35A, 251-278. (Originally presented as a paper at the meeting of the Experimental Psychology Society, Oxford, England, July 1963)
- Drury, C. G. (1975). Application of Fitts' law to foot-pedal design. *Human Factors*, 17, 368-373.
- English, W. K., Engelbart, D. C., & Berman, M. I. (1967). Display-selection techniques for text manipulation. *IEEE Transactions on Human Factors in Electronics*, HFE-8, 21-31.
- Epps, B. W. (1986). Comparison of six cursor control devices based on Fitts' law models. *Proceedings of the 30th Annual Meeting of the Human Factors Society*, 327-331. Santa Monica, CA: Human Factors Society.
- Evans, K. B., Tanner, P. P., & Wein, M. (1981). Tablet-based valuator that provide one, two, or three degrees of freedom. *Computer Graphics*, 15(3), 91-97.
- Ewing, J., Niehrabanzad, S., Sheck, S., Ostroff, D., & Shneiderman, P. (1986). An experimental comparison of a mouse and arrow-jump keys for an interactive encyclopedia. *International Journal of Man-Machine Studies*, 24, 29-45.
- Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47, 381-391.
- Fitts, P. M., & Peterson, J. R. (1964). Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67, 103-112.
- Fitts, P. M., & Radford, B. K. (1966). Information capacity of discrete motor responses under different cognitive sets. *Journal of Experimental Psychology*, 71, 475-482.
- Flowers, K. A. (1976). Visual "closed-loop" and "open-loop" characteristics of voluntary movement in patients with Parkinsonism and intention tremor. *Brain*, 99, 269-310.
- Foley, J. D. (1987, October). Interfaces for advanced computing. *Scientific American*, 257(4), pp. 127-135.
- Gan, K-C., & Hoffmann, E. R. (1988). Geometrical conditions for ballistic and visually controlled movements. *Ergonomics*, 31, 829-839.
- Gillan, D. J., Holden, K., Adam, S., Rudisill, M., & Magee, L. (1990). How does Fitts' law fit pointing and dragging? *Proceedings of the CHI '90 Conference on Human Factors in Computing Systems*, 227-234. New York: ACM.
- Glass, G. V., & Hopkins, K. D. (1984). *Statistical methods in education and psychology* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.
- Glencross, D., & Barrett, N. (1983). Programming precision in repetitive tapping. *Journal of Motor Behavior*, 15, 191-200.
- Glencross, D. J., & Barrett, N. (1989). Discrete movements. In D. H. Holding (Ed.), *Human skills: Studies in human performance* (2nd ed., pp. 107-146). New York: Wiley.
- Goldman, S. (1953). *Information theory*. New York: Prentice-Hall.
- Goodwin, N. C. (1975). Cursor positioning on an electronic display using lightpen, lightgun, or keyboard for three basic tasks. *Human Factors*, 17, 289-295.

- Gould, J. D., Lewis, C., & Barnes, V. (1985). Effects of cursor speed on text-editing. *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems*, 7-10. New York: ACM.
- Greenstein, J. S., & Arnaut, L. Y. (1988). Input devices. In M. Helander (Ed.), *Handbook of human-computer interaction* (pp. 495-519). Amsterdam: Elsevier.
- Haller, R., Mutschler, H., & Voss, M. (1984). Comparison of input devices for correction of typing errors in office systems. In B. Shackel (Ed.), *Human-Computer Interaction - INTERACT '84* (pp. 177-182). Amsterdam: Elsevier.
- Hancock, P. A., & Newell, K. M. (1985). The movement speed-accuracy relationship in space-time. In H. Heuer, U. Kleinbeck, & K.-H. Schmidt (Eds.), *Motor behavior: Programming, control, and acquisition* (pp. 153-188). New York: Springer-Verlag.
- Hancock, W. M., Langolf, G., & Clark, D. O. (1973). Development of standard data for stereoscopic microscopic work. *AIIE Transactions*, 5(2), 113-118.
- Hartzell, E. J., Dunbar, S., Beveridge, R., & Cortilla, R. (1983). Helicopter pilot response latency as a function of the spatial arrangement of instruments and controls. In F. L. George (Ed.), *Proceedings of the 18th Annual Conference on Manual Control* (pp. 345-364). Dayton, OH: Flight Dynamics Laboratories, Airforce Wright Aeronautical Laboratories.
- Hick, W. E. (1952). On the rate of gain of information. *Quarterly Journal of Experimental Psychology*, 4, 11-26.
- Hyman, R. (1953). Stimulus information as a determinant of reaction time. *Journal of Experimental Psychology*, 45, 188-196.
- Jagacinski, R. J. (1989). Target acquisition: Performance measures, process models, and design implications. In G. R. McMillan, D. Beevis, E. Salas, M. H. Strub, R. Sutton, & L. van Breda (Eds.), *Applications on human performance models to system design* (pp. 135-149). New York: Plenum.
- Jagacinski, R. J., & Monk, D. L. (1985). Fitts' law in two dimensions with hand and head movements. *Journal of Motor Behavior*, 17, 77-95.
- Jagacinski, R. J., Repperger, D. W., Moran, M. S., Ward, S. L., & Glass, B. (1980). Fitts' law and the microstructure of rapid discrete movements. *Journal of Experimental Psychology: Human Perception and Performance*, 6, 309-320.
- Jagacinski, R. J., Repperger, D. W., Ward, S. L., & Moran, M. S. (1980). A test of Fitts' law with moving targets. *Human Factors*, 22, 225-233.
- Jones, T. (1989). Effect of computer pointing devices on children's processing rate. *Perceptual and Motor Skills*, 69, 1259-1263.
- Kantowitz, B. H., & Elvers, G. C. (1988). Fitts' law with an isometric controller: Effects of order of control and control-display gain. *Journal of Motor Behavior*, 20, 53-66.
- Karat, J., McDonald, J. E., & Anderson, M. (1984). A comparison of selection techniques: Touch panel, mouse and keyboard. In B. Shackel (Ed.), *Human-Computer Interaction - INTERACT '84* (pp. 189-193). Amsterdam: Elsevier.
- Kay, H. (1960). Channel capacity and skilled performance. In F. A. Geldard (Ed.), *Defence psychology* (pp. 161-169). London: Pergamon.
- Keele, S. W. (1968). Movement control in skilled motor performance. *Psychological Bulletin*, 70, 387-403.
- Keele, S. W. (1973). *Attention and human performance*. Pacific Palisades, CA: Goodyear Publishing.
- Keele, S. W., & Posner, M. I. (1968). Processing of visual feedback in rapid movements. *Journal of Experimental Psychology*, 77, 155-158.

- Kerr, B. (1975). Processing demands during mental operations. *Journal of Motor Behavior*, 7, 15-25.
- Kerr, B. A., & Langolf, G. D. (1977). Speed of aiming movements. *Quarterly Journal of Experimental Psychology*, 29, 475-481.
- Kerr, R. (1973). Movement time in an underwater environment. *Journal of Motor Behavior*, 5, 175-178.
- Kerr, R. (1978). Diving, adaptation, and Fitts' law. *Journal of Motor Behavior*, 10, 255-260.
- Klapp, S. T. (1975). Feedback versus motor programming in the control of aimed movements. *Journal of Experimental Psychology*, 104, 147-153.
- Knight, A. A., & Dagnall, P. R. (1967). Precision in movements. *Ergonomics*, 10, 321-330.
- Kvålseth, T. O. (1977). A generalized model of temporal motor control subject to movement constraints. *Ergonomics*, 20, 41-50.
- Kvålseth, T. O. (1980). An alternative to Fitts' law. *Bulletin of the Psychonomic Society*, 16, 371-373.
- Kvålseth, T. O. (1981). Information capacity of two-dimensional human motor responses. *Ergonomics*, 24, 573-575.
- Langolf, G. D., Chaffin, D. B., & Foulke, J. A. (1976). An investigation of Fitts' law using a wide range of movement amplitudes. *Journal of Motor Behavior*, 8, 113-128.
- Langolf, G., & Hancock, W. M. (1975). Human performance times in microscope work. *AIIE Transactions*, 7(2), 110-117.
- MacKenzie, I. S. (1989). A note on the information-theoretic basis for Fitts' law. *Journal of Motor Behavior*, 21, 323-330.
- MacKenzie, I. S., & Buxton, W. (in press). Extending Fitts' law to two-dimensional tasks. *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems*. New York: ACM.
- MacKenzie, I. S., Sellen, A., & Buxton, W. (1991). A comparison of input devices in elemental pointing and dragging tasks. *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems*, 161-166. New York: ACM.
- Megaw, E. D. (1975). Fitts tapping revisited. *Journal of Human Movement Studies*, 1, 163-171.
- Mehr, M. H., & Mehr, E. (1972). Manual digital positioning in two axes: A comparison of joystick and trackball controls. *Proceedings of the 16th Annual Meeting of the Human Factors Society*, 110-116. Santa Monica, CA: Human Factors Society.
- Meyer, D. E., Abrams, R. A., Kornblum, S., Wright, C. E., & Smith, J. E. K. (1988). Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, 95, 340-370.
- Meyer, D. E., Smith, J. E. K., Kornblum, S., Abrams, R. A., & Wright, C. E. (1990). Speed-accuracy tradeoffs in aimed movements: Toward a theory of rapid voluntary action. In M. Jeannerod (Ed.), *Attention and performance XIII* (pp. 173-226). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Meyer, D. E., Smith, J. E. K., & Wright, C. E. (1982). Models for the speed and accuracy of aimed limb movements. *Psychological Review*, 89, 449-482.
- Miller, G. A. (1953). What is information measurement? *American Psychologist*, 8, 3-11.
- Milner, N. P. (1988). A review of human performance and preferences with different input devices to computer systems. In D. Jones & R. Winder (Eds.), *People and*
-

- Computers IV: Proceedings of the Fourth Conference of the British Computer Society—Human-Computer Interaction Group* (pp. 341-362). Cambridge, England: Cambridge University Press.
- Newell, A., & Card, S. K. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, 1, 209-242.
- Pew, R. W. (1974). Human perceptual-motor performance. In B. H. Kantowitz (Ed.), *Human information processing: Tutorials in performance and cognition* (pp. 1-39). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.
- Pew, R. W., & Baron, S. (1983). Perspectives on human performance modelling. *Automatica*, 19, 663-676.
- Pierce, J. R. (1961). *Symbols, signals, and noise: The nature and process of communication*. New York: Harper & Brothers.
- Radwin, R. G., Vanderheiden, G. C., & Lin, M-L. (1990). A method for evaluating head-controlled input devices using Fitts' law. *Human Factors*, 32, 423-438.
- Reza, F. M. (1961). *An introduction to information theory*. New York: McGraw-Hill.
- Rouse, W. B. (1980). *Systems engineering models of human-computer interaction*. New York: Elsevier.
- Salmoni, A. W. (1983). A descriptive analysis of children performing Fitts' reciprocal tapping task. *Journal of Human Movement Studies*, 9, 81-95.
- Salmoni, A. W., & McIlwain, J. S. (1979). Fitts' reciprocal tapping task: A measure of motor capacity? *Perceptual and Motor Skills*, 49, 403-413.
- Schmidt, R. A. (1988). *Motor control and learning* (2nd ed.). Champaign, IL: Human Kinetics Publishers.
- Schmidt, R. A., Zelaznik, H. N., & Frank, J. S. (1978). Sources of inaccuracy in rapid movement. In G. E. Stelmach (Ed.), *Information processing in motor control and learning* (pp. 183-203). New York: Academic.
- Schmidt, R. A., Zelaznik, H. N., Hawkins, B., Frank, J. S., & Quinn, J. T., Jr. (1979). Motor-output variability: A theory for the accuracy of rapid motor acts. *Psychological Review*, 86, 415-451.
- Shannon, C. E., & Weaver, W. (1949). *The mathematical theory of communication*. Urbana: University of Illinois Press.
- Sheridan, M. R. (1979). A reappraisal of Fitts' law. *Journal of Motor Behavior*, 11, 179-188.
- Sheridan, T. B., & Ferrell, W. R. (1963). Remote manipulator control with transmission delay. *IEEE Transactions on Human Factors in Electronics*, 4, 25-29.
- Sperling, B. B., & Tullis, T. S. (1988). Are you a better "mouser" or "trackballer"? A comparison of cursor-positioning performance. *SIGCHI Bulletin*, 19(3), 77-81.
- SPSS⁺ release 3.1. (1990). [Computer program]. New York: SPSS.
- Sugden, D. A. (1980). Movement speed in children. *Journal of Motor Behavior*, 12, 125-132.
- Tello, E. R. (1988, September). Between man and machine. *Byte*, pp. 288-293.
- Thomas, C., & Milan, S. (1987). Which input device should be used with interactive video? In B. Shackel (Ed.), *Human-Computer Interaction—INTERACT '87* (pp. 587-592). Amsterdam: Elsevier.
- Vince, M. A. (1948). Corrective movements in a pursuit task. *Quarterly Journal of Experimental Psychology*, 1, 85-103.
- Wade, M. G., Newell, K. M., & Wallace, S. A. (1978). Decision time and movement

- time as a function of response complexity in retarded persons. *American Journal of Mental Deficiency*, 83, 135-144.
- Wallace, S. A., & Newell, K. M. (1983). Visual control of discrete aiming movements. *Quarterly Journal of Experimental Psychology*, 35A, 311-321.
- Wallace, S. A., Newell, K. M., & Wade, M. G. (1978). Decision and response times as a function of movement difficulty in preschool children. *Child Development*, 49, 509-512.
- Ware, C., & Baxter, C. (1989). Bat brushes: On the uses of six position and orientation parameters in a paint program. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems*, 155-166. New York: ACM.
- Ware, C., & Mikaelian, H. H. (1987). An evaluation of an eye tracker as a device for computer input. *Proceedings of the CHI+GI '87 Conference on Human Factors in Computing Systems and Graphics Interface*, 183-188. New York: ACM.
- Welford, A. T. (1960). The measurement of sensory-motor performance: Survey and reappraisal of twelve years' progress. *Ergonomics*, 3, 189-230.
- Welford, A. T. (1968). *Fundamentals of skill*. London: Methuen.
- Welford, A. T., Norris, A. H., & Shock, N. W. (1969). Speed and accuracy of movement and their changes with age. *Acta Psychologica*, 30, 3-15.
- Woodworth, R. S. (1899). The accuracy of voluntary movements [Monograph supplement]. *Psychological Review*, 3(2, Whole No. 13).
- Wright, C. E., & Meyer, D. E. (1983). Conditions for a linear speed-accuracy trade-off in aimed movements. *Quarterly Journal of Experimental Psychology*, 35A, 279-296.
- Zelaznik, H. N., Mone, S., McCabe, G. P., & Thaman, C. (1988). Role of temporal and spatial precision in determining the nature of the speed-accuracy trade-off in aimed-hand movements. *Journal of Experimental Psychology: Human Perception and Performance*, 14, 221-230.
- Zimmerman, T. G., Lanier, J., Blanchard, C., Bryson, S., & Harvill, Y. (1987). A hand gesture interface device. *Proceedings of the CHI+GI '87 Conference on Human Factors in Computing Systems and Graphics Interface*, 189-192. New York: ACM.

HCI Editorial Record. First manuscript received January 1990. Revisions received October 29, 1990, and March 22, 1991. Accepted by Stuart Card. — *Editor*

COMPUTER ASSISTED INSTRUCTION

AND INTELLIGENT TUTORING SYSTEMS

Shared Goals and Complementary Approaches

edited by

Jill H. Larkin and Ruth W. Chabay

Carnegie Mellon University and

Center for Design of Educational Computing

A VOLUME IN THE TECHNOLOGY AND EDUCATION SERIES

The fields of computer-assisted instruction and intelligent tutoring systems have had few vehicles for sharing ideas or programs. Different backgrounds and settings meant reading different journals and attending different conferences. The purpose of this book is to foster a mutual understanding of shared issues and contemporary approaches so as to further powerful educational applications of computing. It is unique in drawing on both the intelligent tutoring systems and computer assisted instruction communities.

Each expert contributor provides an in-depth discussion of current work, focusing on instructional programs -- their design, use, and evaluation. The editors and authors have made extensive efforts to make each chapter clear and readable to both communities.

Contents: S. Dugdale, The Design of Computer-Based Mathematics Instruction. G.R. Culley, From Syntax to Semantics in Foreign Language CAI. A.T. Corbett, J.R. Anderson, LISP Intelligent Tutoring System: Research in Skill Acquisition. B.J. Reiser, D.Y. Kimberg, M.C. Lovett, M. Ranney, Knowledge Representation and Explanation in GIL, An Intelligent Tutor for Programming. R.W. Chabay, B.A. Sherwood, A Practical Guide for the Creation of Educational Software. G. Brackett, Realizing the Revolution: A Brief Case Study. A. Lesgold, S. LaJoie, M. Bunzo, G. Eggan, SHERLOCK: A Coached Practice Environment for an Electronics Troubleshooting Job. W. Sack, E. Soloway, From PROUST to CHIRON.
0-8058-0232-0 [cloth] / 1992 / 288pp. / \$49.95
0-8058-0233-9 [paper] / \$24.95

Lawrence Erlbaum Associates, Inc.

365 Broadway, Hillsdale, NJ 07642

201/666-4110 FAX 201/666-2394

Call toll-free to order: 1-800-9-BOOKS-9

...9am to 5pm EST only.



Subscription Order Form

Please ☐ enter ☐ renew my subscription to

HUMAN-COMPUTER INTERACTION

Volume 7, 1992, Quarterly

Subscription prices per volume:

Individual: ☐ \$39.00 (US/Canada) ☐ \$64.00 (All Other Countries)
Institution: ☐ \$145.00 (US/Canada) ☐ \$170.00 (All Other Countries)

Subscriptions are entered on a calendar-year basis only and must be prepaid in US currency directly to the publisher. Individual orders must be prepaid by personal check, money order, or credit card. Institutional checks for individual orders will not be accepted. Offer Expires 12/31/92.

☐ **Payment Enclosed**

Total Amount Enclosed \$ _____

☐ **Charge My Credit Card**

☐ VISA ☐ MasterCard ☐ AMEX ☐ Discover

Exp. Date _____ \ _____

Card Number _____

Signature _____

(Credit card orders cannot be processed without your signature.)

Please print clearly to ensure proper delivery.

Name _____

Address _____

City _____ State _____ Zip+4 _____

Prices are subject to change without notice.

Lawrence Erlbaum Associates, Inc.
Journal Subscription Department
365 Broadway, Hillsdale, New Jersey 07642
(201) 666-4110 FAX (201) 666-2394

INTELLIGENCE: RECONCEPTUALISATION AND MEASUREMENT

edited by
Helga A.H. Rowe
Australian Council for Educational Research

As reform in all sectors of education continues, it is becoming increasingly important that we develop a rich understanding of what "intelligence" is, and how it can be improved. Reflecting current views on the manifestation, development, and assessment of human intelligence, this volume addresses a rich diversity of theoretical, methodological, and applied issues -- a number of which have not been raised previously. The contributors to this collection -- highly regarded experts from various countries -- propose perspectives for future research, their intent being not so much to predict the future, but to help shape it.

Contents: Foreword. **H.A.H. Rowe**, Introduction: Paradigm and Context. **M. Richelle**, Reconciling Views on Intelligence. **K. Raahel**, Is the High IQ Person Really in Trouble? **R. Glaser**, Intelligence as an Expression of Acquired Knowledge. **J.B. Biggs, K. Collis**, Multimodal Learning and the Quality of Intelligent Behaviour. **S.R. Goldman, J.W. Pellegrino**, Cognitive Developmental Perspectives in Intelligence. **L. Stankov**, The Effects of Practice and Training on Human Abilities. **J.D. Crawford**, Intelligence Task Complexity and the Distinction Between Automatic and Effortful Mental Processing. **D.L. Robinson**, On the Neurology of Intelligence and Intelligence Factors. **J.P. Das, R.F. Jarman**, Cognitive Integration: An Alternative Model for Intelligence. **R.J. Sternberg**, Theory-Based Testing of Intellectual Abilities: Rationale for the Sternberg Triarchic Abilities Test. **J.W. Pellegrino**, Cognitive Models for Understanding and Assessing Spatial Abilities. **R. Kluwe, C. Misiak, H. Halder**, The Control of Complex Systems and Performance in Intelligence Tests. **A. DiSessa**, Transforming Intelligence With Computers. **A.J. Cropley**, Improving Intelligence: Fostering Creativity in Everyday Settings. **K. Harris**, Intelligence, Economics and Schooling.

0-8058-0942-2 / 1991 / 312pp. / \$49.95

Lawrence Erlbaum Associates, Inc.
365 Broadway, Hillsdale, NJ 07642
201/666-4110 FAX 201/666-2394

Call toll-free to order: 1-800-9-BOOKS-9
9am to 5pm EST only.

LEXICAL ACQUISITION

Exploiting On-Line Resources to Build a Lexicon

edited by

Uri Zernik

General Electric Research and Development Center

On-line information -- and free text in particular -- has emerged as a major, yet unexploited, resource available in raw form. Available, but not accessible. The lexicon provides the major key for enabling accessibility to on-line text.

The expert contributors to this book explore the range of possibilities for the generation of extensive lexicons. In so doing, they investigate the use of existing on-line dictionaries and thesauri, and explain how lexicons can be acquired from the corpus -- the text under investigation -- itself. Leading researchers in four related fields offer the latest investigations: computational linguists cover the natural language processing aspect; statisticians point out the issues involved in the use of massive data; experts discuss the limitations of current technology; and lexicographers share their experience in the design of the traditional dictionaries.

Contents: U. Zernik, Introduction. **Part I: *Lexical Senses*.** P. Jacobs, Making Sense of Lexical Acquisition. R. Krovetz, Lexical Acquisition and Information Retrieval. B. Slaton, Using Context for Sense Preference. U. Zernik, Tagging Word Sense In Corpus. **Part II: *Lexical Statistics*.** K. Church, W. Gale, P. Hanks, D. Hindle, Using Statistics in Lexical Analysis. F. Smadja, Macrocoding the Lexicon with Co-Occurrence Knowledge. N. Calzolari, Lexical Databases and Textual Corpora: Perspectives of Integration for a Lexical Knowledge-Base. **Part III: *Lexical Representation*.** R. Beckwith, C. Fellbaum, D. Gross, G. Miller, WordNet: A Lexical Database Organized on Psycholinguistic Principles. B. Atkins, B. Levin, Admitting Impediments. B. Dorr, Conceptual Basis of the Lexicon in Machine Translation. M. Dyer, Lexical Acquisition Through Symbol Recirculation. **Part IV: *Lexical Semantics*.** P. Velardi, Acquiring a Semantic Lexicon for Natural Language Processing. L. Braden-Harder, W. Zadrozny, Lexicons for Broad Coverage Semantics. J. Martin, Representing and Acquiring Metaphor-Based Polysemy.

0-8058-0829-9 [cloth] / 1991 / 440pp. / \$69.95

0-8058-1127-3 [paper] / \$34.50

Lawrence Erlbaum Associates, Inc.

365 Broadway, Hillsdale, NJ 07642

201/666-4110 FAX 201/666-2394



Call toll-free to order: 1-800-9-BOOKS-9...9am to 5pm EST only.

ARTIFICIAL BELIEVERS

Afzal Ballim

Institut Dalle Molle ISSCO, University of Geneva

Yorick Wilks

*Computing Research Laboratory
New Mexico State University*

Modeling of individual beliefs is essential to the computer understanding of natural languages: phenomena at all levels -- syntactic, semantic, and pragmatic -- cannot be fully analyzed in the absence of models of a hearer and of the hearer's model of other believers. This book provides a prolog program, *Viewgen*, that maintains belief structures about the world and other believers, and is able to ascribe beliefs to others without direct evidence by using a form of default reasoning. The authors contend that a plausible model such as this can -- in the best cognitive science tradition -- shed light on the long-standing philosophical problem of what belief is.

The issues presented here will be of considerable interest to an informed general reader as well as those with a background in any of the disciplines that make up what is now called cognitive science including: philosophy, linguistics, psychology, neuropsychology, as well as AI itself.

Contents: Introduction. Preliminaries on the Nature of Belief. Belief Ascription. Experiments in Belief. Global Issues: Reasoning with Viewpoints. Further Extensions and Speculations.

0-8058-0453-6 / 1991 / 288pp. / \$45.00

Special Prepaid Offer! \$29.95

No further discounts apply.

Lawrence Erlbaum Associates, Inc.

365 Broadway, Hillsdale, NJ 07642

201/666-4110 FAX 201/666-2394

Call toll-free to order: 1-800-9-BOOKS-9
9am to 5pm EST only.



INFORMATION FOR AUTHORS

Human-Computer Interaction is an interdisciplinary journal concerned with theoretical, empirical, and methodological issues of (1) user science and (2) computer system design as it affects the user. The goal of *HCI* is to be a high-quality journal coalescing the best research and design work from diverse fields into a distinct new field of human-computer interaction.

User Science. *HCI* seeks to foster a scientific understanding of the behavior of computer users, especially the cognitive aspects. "Users" include both programmers and non-programmers and both experts and novices. *HCI* is concerned not only with the individual user and with small working groups of users, but also with the larger social and organizational context of the user community. Theoretical papers should deal with psychological models of user learning or performance or with social models of the user community. Empirical papers may range from controlled laboratory experimentation to field observation. Methodological papers should be concerned with such issues as how to analyze tasks and how to discover the structure of user behavior.

System Design. *HCI* seeks to foster rational discussion of and methods for the design of new computer systems and the evaluation of existing systems. *HCI* is interested in user-interface design techniques, including the incorporation of intelligence into the interface. *HCI* is also concerned with the process of design (i.e., not only the *what*, but also the *how* and the *why* of design). Theoretical papers should deal with the design principles underlying a particular system or class of systems, or with the abstract structure and process of human-computer interaction. Empirical papers may assess existing or novel interaction techniques, or examine the design process itself. Methodological papers should be concerned with the application of design principles, the rationalization of design alternatives, or the role of empirical methods in the design process.

HCI is interested in original papers of high quality and broad relevance that are within the scope of the above thematic framework. Papers may cover any domain in which computers are involved—such as programming, office systems, database systems, instructional and training systems, games, design systems, editing and authoring systems, and other interactive computer tools. Because *HCI* is an interdisciplinary journal, all papers must have both scientific content or implications *and* practical relevance as to how systems should be designed or how they are actually used. *HCI* favors substantial papers dealing with substantial pieces of research and/or design over smaller or narrower papers focusing on issues of only specialized interest. Review papers are welcomed but are expected to develop new conceptual views or codifications of existing material.

Manuscript Style. Any readable, complete manuscript format is acceptable for initial reviewing. However, manuscripts will be expected to conform to the *HCI* journal style (best understood by looking at a recent issue of *HCI*). In general, final manuscripts should be prepared in accordance with the *Publication Manual of the American Psychological Association*, Third Edition (1200 Seventeenth Street, N.W., Washington, DC 20036). The specific style rules for final *HCI* manuscripts are available from the Editor.

Submission. To submit a manuscript to *HCI*, send *five* copies to:

Thomas P. Moran
Editor, *Human-Computer Interaction*
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

HUMAN-COMPUTER INTERACTION

Contents of Volume 7, Number 1

1992

ARTICLES

- Temporal Aspects of Tasks in the User
Action Notation1**
H. Rex Hartson and Philip D. Gray
- Inferring Graphical Procedures:
The Compleat Metamouse47**
*David L. Maulsby, Ian H. Witten,
Kenneth A. Kittlitz, and Valerio G. Franceschin*
- Fitts' Law as a Research and Design Tool in
Human-Computer Interaction91**
I. Scott MacKenzie
-